Performance of Network Intrusion Detection and Prevention Systems in High-speed Environments

Bulajoul, Waleed

Award date:
2017

Awarding institution:
Coventry University

Performance of Network Intrusion Detection and Prevention Systems in High-speed Environments

By

Waleed A A Ahmed Bul'ajoul

Your Award (PhD)

May 2017

Performance of Network Intrusion Detection and Prevention Systems in High-speed Environments

By

Waleed A A Ahmed Bul'ajoul

May 2017

A thesis submitted in partial fulfilment of the University's requirements for the Degree of Doctor of Philosophy

Abstract

Due to the numerous and increasingly malicious attacks on computer networks and systems, current security tools are often not enough to resolve the issues related to illegal users, reliability, and to provide robust network security. Recent research has indicated that although network security has developed, a major concern about an increase in illegal intrusions is still occurring. Addressing security on every occasion or in every place is a really important and sensitive matter for many users, businesses, governments and enterprises. A Network Intrusion Detection and Prevention System (NIDPS) is one of the most tested, reliable, and strongest forms of technology used to sniff out network packets, monitor incoming and outgoing network traffic, and identify the unauthorised usage and mishandling of computer system networks. It can provide a better understanding of the things that are really happening on the network. In addition, an NIDPS has the potential to detect, prevent, and report any evidence of attacks and malicious traffic. It is critical to implement an NIDPS in a computer network that has high traffic and high-speed connectivity. This thesis presents an investigation, involving literature review and intensive experiments, which shows that current NIDPSs have several shortcomings such as they are incapable to detect or prevent the rising attacks and threats to high-speed environments, such as flood attacks (UDP, TCP, ICMP and HTTP) or Denial and Distributed Denial of Service attacks (DoS/DDoS), because the main purpose of these types of attacks is basically to send heavy traffic to systems at high-speed to stop or slow down performance. To investigate the status of NIDPS performance and test the capability of NIDPS analysis, detection, and prevention modes when exposed to malicious attacks that come through high-load and high-speed traffic, a prototype network has been designed. The prototype consisted of virtual and physical stations including six (6) PCs and three (3) switches (i.e two layer 2 switches and 1 layer 3 switch). Several tools were used to carry out the research experiments, implementation and evaluation. The research presents a study using Snort NIDPS open source software. It shows that NIDPS performance can be weak in the face of high-speed and high-load traffic in terms of packet drops, and outstanding packets without analysis and failing to detect/prevent unwanted traffic. The research has designed a novel QoS architecture to increase the analytical, detection, and prevention performance of NIDPS when deployed in high-speed networks. It has proposed and evaluated a solution using a novel QoS configuration in a multi-layer switch to organise and improve network traffic performance in order to reduce the packets dropped and then uses parallel techniques to increase packet processing speed. The novel architecture was tested under different traffic speeds, types, and tasks. The experimental results show that the novel architecture improves network and NIDPS performance.

Table of Contents

CHAPTER 1: INTRODUCTION

1.1 Introduction

The research is introduced in this chapter, including the background, motivation and purpose, leading to a problem statement and introduction of research questions. The research aims and objectives are stated and an overview of the research methodology and approach is also provided. An analysis of the research relative to the state of the art and a summary of its original contributions are included. Finally, the structure of the thesis is described.

1.2 Introduction to the Research

The drastic growth of various network and wireless systems-based malicious attacks has rendered conventional security tools, such as firewalls and anti-virus programs, insufficient to provide integral, reliable and secure free networks. Intrusion Detection and Prevention Systems (IDPS) offer strong, reliable technology that monitor inbound and outbound network traffic to detect illegal usage and corruption of systems (Scarfone and Mell 2007, Bul'ajoul, James and Pannu 2013, Bul'ajoul, James and Pannu 2015 and Kenkre, Pai and Colaco 2015). IDPS also identify the activity of malicious attackers. Several computer network systems are incompetent to stop computer network terrorisations such as overflow attacks, including Transmission Control Protocol (TCP) flood, User Datagram Protocol (UDP) flood and Hypertext Transfer Protocol (HTTP) flood. In addition, Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks disturb numerous systems, and the influence of such attacks is severe and serious. The key technique of these attacks is to send traffic at high-speed and high-volume to a network system address, which can be stopped or slowed down the performance by taking advantages of system vulnerabilities such as misconfigurations and software bugs (Bul'ajoul, James and Pannu 2015, Goel and Mehtre 2016, Ordi et al. 2015, Wang et al. 2015 and Van der et al. 2015).

IDPS are capable of monitoring, identifying and reporting evidence of malicious activities and attacks such as DoS and DDoS, unauthorized log-ins, privilege escalation, illegitimate access and modification of data and data-driven attacks. Therefore, an IDPS's sniffing mechanism can be applied at a network gateway to provide offered valuable information about traffic and packet types to security professionals. IDPS are widely used to reduce the risks of attacks.

This research is based on Network Intrusion Detection and Prevention Systems (NIDPS). NIDPS involve software in which incoming and outgoing individual and/or network traffic is

monitored. An open-source IDPS software has been used in this research, which acts as a Network Intrusion Prevention System (NIPS) and Network Intrusion Detection System (NIDS), and can therefore be considered a NIDPS. Snort NIDPS open source is quite famous within the research community. It is the most widely deployed IDPS worldwide (Akhtar, Matta and Wang 2015, Sadhukhan, Mallari and Yadav 2015 and Samani and Karamta 2016). Snort NIDPS has three modes: Network IDPS mode; Host IDPS mode; and Hybrid IDPS mode. This research used Network Sniffing, Network Detection and Network Prevention configurations within the NIDPS mode. Many studies have used Snort IDPS to observe and monitor high-speed levels of network environments and detect high-speed traffic attacks, such as DoS, DDoS and flood attacks, by developing and designing new rules (Buchanan et al. 2011, Saboor, Akhlaq and Aslam 2013, Khamphakdee, Benjamas and Saiyod 2014, Khamphakdee, Benjamas and Saiyod 2015, Bul'ajoul, James and Pannu 2015 and Goel and Mehtre 2016). The research demonstrates the weaknesses of the NIDPS, namely its incapability to process various traffic at high-speed and heavy volumes of packets, and its inability to detect or prevent unwanted traffic that might attack the system. The research proposes a solution to improve NIDPS performance. This study shows how a novel configuration of QoS (Quality of Service) in the Layer 3 network switch combined with the introduction of parallel technologies can increase the efficiency and effectiveness of NIDPS platforms.

1.3 Motivation and Purpose

Information technology (IT) influences almost every aspect of modern life. Today, various devices are available to meet users' requirements such as high machine processor speed, quick network responses and reliable security. Alongside our increasing dependence on IT, there has unfortunately been a rise in security incidents (Arbor Networks 2015). Threats and attacks may range from stealing personal information from a laptop or network server to stealing the most top-secret information stored on a Security Intelligence Service (SIS). Furthermore, hackers can snoop on users' online purchases by eavesdropping on their credit card details, or, even more dangerous, a damaged web resource can cause failures in the movement of air traffic. Multi-faceted attacks and threats have made the implementation of security systems more challenging. Hackers have evolved along with the sophistication of the IT industry. For example, hackers exploit the developments in computer processors and network speeds (multi-core and cloud environment technologies) to increase the volume and speed of malicious traffic that might constitute a DoS or DDoS attack (Cheung et al. 2009, Gao and Xiao 2012, Wang et al. 2015, Chauhan and Prasad 2015, Malina, Dzurenda and Hajny 2015, Bukac and Matyas, 2015 and Samani and Karamta, 2016). Network security is therefore extremely important and has developed into an industry aimed at improving applications and hardware platforms to identify and stop network threats.

One of the most established concepts in the information security is a defence-in-depth approach which utilised a multi-layered structural design in which firewalls, vulnerability assessment tools (anti-viruses and worms) and IDPS are employed to prevent any hostile endeavours on network systems and servers. The NIDPS has been designed to serve as the last point of defence in the network architecture. The name "NIDPS" is derived from the fact that the system monitors the network traffic from the viewpoint of where it is installed. NIDPS monitor the transportation of network traffic for any malicious and uncomfortable activities and create alerts when operating in detection mode or block packet alerts when operating in prevention mode. NIDPS can be used to defend and protect network strategic segments and monitor a specific part of a network or system. NIDPS can also be placed in the external border of the network architecture to defend important parts of the system, such as servers, from any intrusion attempts or malicious attacks (Dave, Trivedi and Mahadevia 2013, Vasudeo, Patil and Kumar 2015and Bul'ajoul, James and Pannu 2015).

The detection and prevention mechanisms of the NIDPS are grounded in observing the comparison of ingress packets (traffic) to any known aggression (attack) through patterns (signature NIDPS mechanism) or identifying unknown malicious patterns from ingress traffic (anomaly NIDPS mechanism). NIDPS are important in that they:

- counter intrusions or malicious attempts to access networks and systems;
- analyse network traffic and identify hackers' targets and techniques; and
- detect or prevent unwanted and malicious traffic.

Open source is the most common category of NIDPS software configured platforms (Akhtar, Matta and Wang 2015 and Sadhukhan, Mallari and Yadav 2015); however, its performance in high-speed networks communication remains a major issue. Irrelevant alerts (false positive alerts) occur, thus creating a more difficult job for system security managers. Moreover, despite claims of increased capabilities and efficient performances by several NIDPS dealers, research has shown that systems lack the required capabilities to monitor and analyse high-speed network traffic (Kenkre, Pai, and Colaco 2015, Li et al. 2015, Kim et al. 2015 and Kizza 2015).

Innovators have created current hardware IDPS to process millions of packets at the same time (Cisco 2016b and Trevisan et al. 2016), but there are limitations in the capability to perform particular software tasks. In addition, limited memory size is a problem for hardware-based NIDPS solutions. Furthermore, hardware-based NIDPS offer a high range of processing speed but are very costly. Software solutions are popular because they are cheaper and offer more flexibility than hardware solutions. This research focuses on open-source software solutions.

Computer network and Internet security face increasing challenges and many companies rely on NIDPS to secure their data sources and systems. The need to ensure that the NIDPS can keep up with the increasing demands as a result of increased network usage, higher speed networks and increased malicious activity, makes this an interesting area of research and motivates this study.

1.4 Problem Statement

The problem addressed by this research is that NIDPS is unable to detect or prevent unwanted packets (traffic) when faced with unexpectedly high volumes of traffic exceeding network interface's speed i.e. 100MBps / 1Gbps. This study investigates the problem and proposes a solution.

1.5 Research Questions

The main research question is:

How can network security architecture be improved to cope with high-speed traffic and high-volume data attacks?

To answer the research question, the following sub-questions were addressed:

1. What are the current performance issues with NIDPS?
2. How can network management and traffic performance be improved through QoS technology?
3. Can parallel technology along with QoS improve the performance of NIDPS?
4. Can a generalised NIDPS architecture be built through the application of QoS and parallel technology for improving security performance?

1.6 Research Methodology and Approach

In this study, a quantitative approach based on experimental analysis has been followed. There are three types of quantitative approach: simulation, experimental, and inferential. The simulation approach is concerned with the construction of an artificial network environment, within which relevant database information and traffic (packets) are generated. The experimental approach involves changing variables in an observed domain and monitoring the effects. The inferential approach uses relationships or characteristics of a population to deduce new findings. The simulation and experimental approaches can overlap in that experimentation can be carried out within a

simulated environment. This study initially uses a quantitative methods experimental approach. Experiments were carried out to analyse the performance of NIDPS under various traffic scenarios with and without improved novel security architecture.

1.7 Identifying State of the Art Technology

Regardless of the accessibility of monitoring tools and security-enforcing software, the understanding of the issues involved in designing a software security solution are difficult and expensive. Although effective techniques can be learned that identify and detect different types of attacks, there are no software approaches and that are realisable globally and that achieve unfailing results for detecting numerous types of high-level and high-speed attacks. The discoveries of cyber-based attacks on network systems continue to be an important and challenging area of research.

Malicious traffic and attacks continue to grow and develop daily. Current security software cannot keep up with this development; it is still unable to handle some high-speed attacks such as DoS, DDoS and flood attacks (TCP, UDP and ICMP). The largest speed attack in the history was carried out against the BBC website. It is DDoS attack. The attack's speed reached up to 602 Gbps (The Hacker News 2016), which is far higher than the speed of current networks and security systems. The previous record was 400Gbps in 2014 (Arbor Networks 2015). Detection of such kind attacks by traditional network security operations alone is almost difficult due to the rise of network traffic speed and volume of data and real-time environment.

This study investigates the concern that current security systems, while continuously developing, are not keeping up with the increasingly high-speed intrusions into computer networks. This study further demonstrates how to improve the methods used by the NIDPS for analysing, detecting, and preventing attacks in network security systems. There are two major areas of concern in computer security: the speed and volume of attacks, and the complexity of multi-stage attacks (De Muila and Ferdinand 2010 and Bul'ajoul, James and Pannu 2015). The study analyses state of the art developments in network switch and parallel technology to provide a solution to the problem of NIDPS not meeting the demands of high-speed and high-volume attacks.

This research focuses on the improvement of a novel and unique infrastructure for NIDPS based on an open-source software which has a benefit of employing the right usage of QoS and parallel technologies to improve attack detection rates and thus contribute to state-of-the-art network security. In addition, the proposed approach enables handling of the most high-speed and severe attacks faced by the Internet and computer networks.

1.8 Research Aim and Objectives

The aim of the research is to develop a solution to the problem of NIDPS that are unable to cope with high-speed and high-volume traffic attacks.

The objectives of this research are to:

- Review the literature to assess weaknesses of NIDPS;
- Investigate a particular NIDPS to see if such weaknesses exist;
- Design an experiment that will enable NIDPS to be tested under high-speed traffic with various network packets;
- Design new architecture based on a novel QoS configuration to improve NIDPS performance;
- Design and develop a parallel implementation of the novel architecture;
- Carry out experiments to test the improved architecture; and
- Evaluate the redesigned NIDPS architecture.

1.9 Original Contribution

This research offers an original contribution, described as follows:

- Development of a new architecture for NIDPS that improves overall network security and performance.

  The architecture is founded on the integration and development of two techniques:

  1. A novel QoS architecture was designed that enhances network traffic performance based on classification and policy methods and then improves overall network management and security.
  2. A high level parallelism was designed for the novel architecture to improve traffic throughput processing with the aim of improving NIDPS throughput performance and reducing NIDPS processor time.

1.10 Thesis Structure

The remaining chapters are organised as the follows:

Chapter 2: Literature Review

This chapter discusses the historical perspective and provides information about the development of threats and attacks; security mechanisms; types of intrusion detection and

prevention technologies and their methodologies; some discussion of open-source IDPS, Snort and Bro; and, finally, related research in parallel technology.

Chapter 3: Methodology and Experimental Design
This chapter explains the methodology and experimental design, including the configuration of Snort NIDPS rules and components. The experimental testbed is described along with its constituent parts and the experiments carried out are listed.

Chapter 4: Exposure of Problem through Experimentation
This chapter exposes the research problem in more detail through implemented experiments. The weakness of current NIDPS technology is shown.

Chapter 5: A Novel Architecture
This chapter presents a solution to the problem of dropped packages and missed attacks. The solution proposed is based on Layer 3 QoS switch technology and parallel processing.

Chapter 6: Evaluation of the Novel Architecture
An evaluation of the proposed solution through a second set of experiments is presented in Chapter 6. The solution is shown to improve NIDPS performance.

Chapter 7: Conclusion, Recommendation and Future work
Finally, Chapter 7 concludes the study and suggests recommendations and further work.

1.11 Research Outputs

In this section publications which have arisen from the research are listed.

1.11.1 Publications arising from the Research

1. Bulajoul, W., James, A., and Pannu, M. (eds.) (2013) *E-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on*. 'Network Intrusion Detection Systems in High-Speed Traffic in Computer Networks': IEEE.

2. Bul'ajoul, W., James, A., and Pannu, M. (2015) 'Improving Network Intrusion Detection System Performance through Quality of Service Configuration and Parallel Technology'. *Journal of Computer and System Sciences* 81 (6), 981-999.

3. Bul'ajoul, W., James, A., Shaikh. S, and Pannu, M. (2016) 'Using Cisco Network Components to Improve NIDPS Performance. *Second International Conference of Networks, Communications, Wireless and Mobile Computing (NCWC 2016).*

4. James, A., Bulajoul, W., Shehu, Y., Li, Y and Obande, G (2017) 'Security Challenges and Solutions for E-Business', *The Institution of Engineering and Technology (IET). (accepted)*

5. Bul'ajoul, W., and James, A. (2017). Intrusion Detection Systems: Management, Technology and Recent Advances. *Nova Science Publishers*. *(The abstract is accepted and chapter under process)*

1.11.2 Posters arising from the Research

1. Bul'ajoul, W., and James, A. 2013. Network Intrusion Detection systems (NIDS) for Multi Attack Scenarios in Computer Networks. *Research Symposium, Coventry University, UK.*

2. Bul'ajoul, W., and James, A. 2014. Intrusion Detection Systems for High-speed Environments. *BCS Symposium, University of Warwick, UK.* (Achieved Award)

3. Bul'ajoul, W., and James, A. 2014.Intrusion Detection Systems for High-speed Environments. *EC Annual Research Symposium, Coventry University, UK.* (Achieved Award)

4. Bul'ajoul, W., and James, A. 2014. Intrusion Detection systems for Multi Attack Scenarios in Computer Networks. *Libyan Higher Education Forum-London, Libyan Embassy, London, UK.* (Achieved Award)

5. Bul'ajoul, W., and James, A. 2014. Network Intrusion Detection Systems for High-speed Traffic in Computer Networks. *PGR Symposium, Coventry University, UK.*

1.11.3 Presentations arising from the Research

1. Bul'ajoul, W., and James, A. 2013. Network Intrusion Detection Systems for High-Speed Traffic in Computer Networks. *2013, IEEE 10th International Conference, Coventry University, UK.*

2. Bul'ajoul, W., and James, A. 2014. Intrusion Detection Systems for High-speed Environments. *EC Annual Research Symposium, Coventry University, UK.*

3. Bul'ajoul, W., and James, A. 2014. Improving Network Intrusion Detection System Performance through Quality of Service Configuration and Parallel Technology. *PGR Symposium, Coventry University, UK.*

4. Bul'ajoul, W., James, A., Shaikh. S, and Pannu, M. (2016) 'Using Cisco Network Components to Improve NIDPS Performance. *Second International Conference of Networks, Communications, Wireless and Mobile Computing (NCWC 2016), Dubai, UAE.*

1.12 Conclusion

This chapter has given a brief background to the research, describing the motivation, problem statement and research questions addressed. It has also briefly described the methodology and set out the aims and objectives. The original contribution has been highlighted and the thesis structure outlined. Finally a list of publications and presentations made in connection with this research has been included in this chapter.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter focuses on the state-of-the-art and literature review of related work. It gives an overview and background about the latest threats and attacks in computer networks, common security mechanisms and approaches including IDPS types and methodologies and discusses advances in research towards improving the performance of NIDPS.

2.2 Threats and Attacks

A company's network plays a vital role in its business projects. Keeping the computer network up-to-date with the latest software and security techniques is essential for success and progress. Reliability and safety are the major concerns in enabling a company to achieve success and boost its progress.

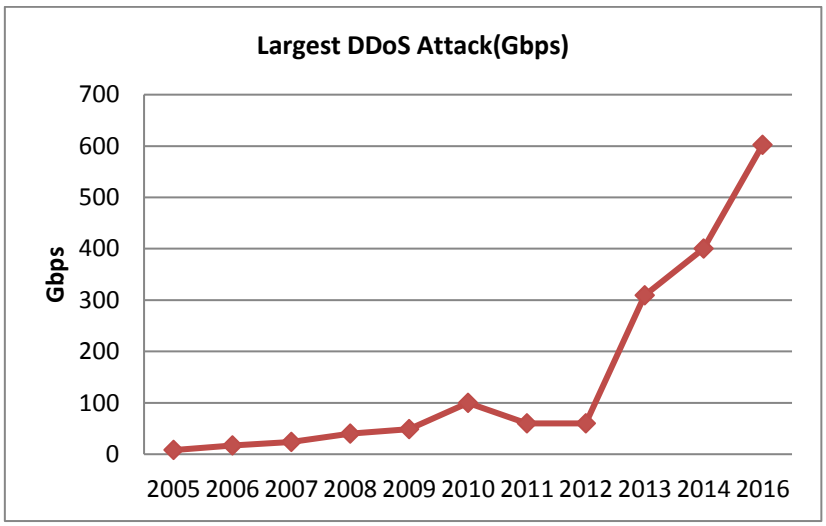


Figure 2. 1: Largest DDoS attack reported by The Hacker News 2016 and Arbor Networks 2015.

However, networks can also be considered a major risk in any business project. Security issues have increased as technology has advanced (Jang-Jaccard and Nepal 2014). Fuchsberger (2013) reported that, according to a survey conducted by the Federal Bureau of Investigation and Crime Scene of Investigation (FBI/CSI), viruses are behind many attacks on business networks. Moreover, Denial of Service (DoS) attacks and unauthorized user access (which can be initiated from external or internal LAN sources) have also increased dramatically. It is also noticeable that nowadays there are

powerful intrusion tools available, allowing hackers to attack networks even if they know little of the software. Attackers can now use several tools simultaneously to achieve an objective.

The 10th Annual Worldwide Infrastructure Security Report and ATLAS 2015 data report (Arbor Networks 2015) reported that the number of Distributed Denial of Service (DDoS) attacks has grown significantly, nearly doubling on a year-to-year basis between 2005 and 2010 (see Figure 2.1). The size of attacks in 2016 increased by over 33 percent compared to the previous year. The largest reported attack by BBC website respondents in 2016 was over 600 Gbps (The Hacker News 2016); the previously largest reported attack size by WISR was recorded at 400 Gbps. Moreover, ATLAS recorded more than 8x number of malicious attacks over 20 Gbps as compared to 2012 in 2013, The largest monitored attack by ATLAS in 2014 was slightly more than that at 324Gbps. Ten years ago the largest monitored attack was 8Gbps. Recent DDoS attacka have utilised networks and internet servers that employ a large number of automated detection and mitigation techniques in order to prevent the misuse of the services. The hackers simply use their script on the server's services to set the bandwidth limit as unlimited and to hide the amendment.

Therefore, security products, such as firewalls, vulnerability assessment tools, antivirus programs, and Network Intrusion Detection and Prevention Systems (NIDPSs), are utilised to reduce the risk of attacks. However, even these measures are not 100 percent effective in protecting networks. One problem is that increase network traffic speed and valoume over its limit, packets can be dropped prior to analysis, detection and prevention (Shiri, Shanmugam and Ideis 2011, Albin and Rowe 2012, Bul'ajoul, James and Pannu 2015 and Kenkre, Pai and Colaco 2015). It is becoming recognised that advantage could be taken of parallel technology such as multi-core technology or distributed systems to overcome the problem of the network traffic rate superseding the rate at which NIDPSs can process incoming data (Jiang et al. 2014, Bul'ajoul, James and Pannu 2015 and Zhang et al. 2015).

Flood attacks, also known as DoS attacks or DDoS attacks, are deliberate over-requesting of network system resources which become rendered unavailable. Statistically, the most common attack is flood attack. Between 2011 and 2014, there was a massive rise in the speed at which this attack occurred, indicating that the security system must be able to perform at similarly attacks's speed (Wei and Xiangliang 2011 and Arbour Networks 2015).

Hardware-based IDPS are more powerful in performance terms than software-based and some companies have opted to use such technology as defence against the rapid development of bandwidth and the speed of attacks (Lesk 2007, TeleGeography Research 2008 and M86 Security 2010). NIDPS become inefficient when attempting to monitor high-speed and volume network traffic and will end

up dropping, outstanding packets and losing alerts, logs, and blockings of malicious packets when the software is implemented as a solution (Whitman et al. 2012, Weaver, Weaver and Farwood 2013:265-292, Hussain, Lalmuanawma and Chhakchhuak 2015 and Bul'ajoul, James and Pannu 2015). However, hardware IDPSs are very costly and are not available in the many organisations and companies. A solution is therefore needed to enhance software approaches in the context of affordable infrastructure. This research focuses on this challenge.

2.3 Security mechanisms and approaches

Security products such as firewalls and antivirus programs are less efficient than NIDPS and have different functionalities. NIDPSs analyse collected information and detect or prevent unwanted traffic to infer more useful results than other security products. The difference between NIDPS and other security products such as antivirus programs is that, while NIDPS require more embedded intelligence than other security products, they analyse gathered information and deduce useful results (Bul'ajoul, James and Pannu 2013, Bul'ajoul, James and Pannu 2015 and Ahmed, Khan and Bashir 2015).

2.3.1 Firewall technology

In order to secure a corporate network or a sub-network, network traffic is usually filtered according to criteria such as origin, destination, protocol and service, typically forwarded from dedicated router to firewall through the network and also could be placed before router. It depends on the network and policy requirements (Khorchani, Halle and Villemaire 2012 and Bul'ajoul, James and Pannu 2015).

The Firewall is a standard security system defence and has become an important part of all network gateways for stopping inbound and outbound intruders from getting access to private/local networks and systems (Beg et al. 2010, Shuo and Quan 2015 and Genge, Graur, and Haller 2015). The functionality of the firewall is based on filtering mechanisms specified by a set of rules, known as a policy, which can protect a system from such attacks. The fundamental function of the firewall is to sort packets according to allow/deny rules, based on header-field information. The basic operation of the firewall is filtering packets passing through specific hosts or network ports, which are usually open in most computer systems. It does not perform deep analysis (malicious code detection in the packet) and treats each packet as an individual entity (Marinova-Boncheva 2007, Beget al. 2010, Whitman et al. 2012:133-164, Trabelsi, Sayed and Zeidan 2012and Shuo and Quan 2015).

The disadvantage of a firewall is that it cannot fully protect an internal network; it is unable to stop internal attacks, outside attacks such as anomaly attacks (unknown signatures attacks), or high-volume attacks from accepted signatures (Kim and Cho 2012, Wang, Zhang and Song 2012, Bul'ajoul, James and Pannu 2013, Bul'ajoul, James and Pannu 2015 and Shuo and Quan 2015). A firewall is just a set of rules such as to allow or deny protocols, ports or an IP address. Today's denials of service (DoS) attacks are too complex for firewall technology, because it cannot distinguish good traffic from DoS attack traffic (Bessis and Rana 2015, Ormazabal et al. 2015 and Chen et al. 2015). However, the firewall provides the benefit of added security to strengthen a network when used in conjunction with an IDPS (Alpcan and Başar 2010:29-33).

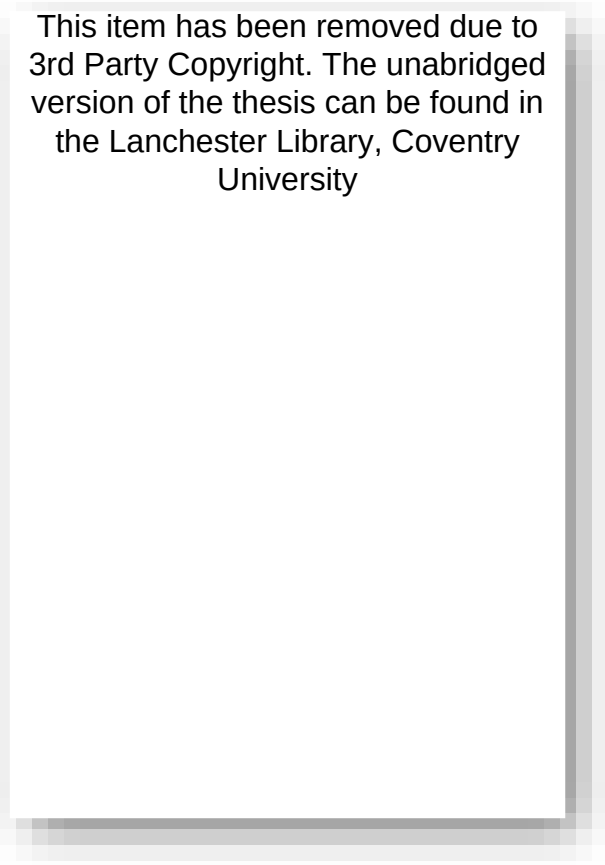


Figure 2. 2: Basic firewall installation.

2.3.2 Anti-virus technology

Computer viruses are programs which cause computer failure and damage computer data. A computer virus poses an immeasurable threat and can be very destructive especially in a network environment. Antivirus programs are software that can be installed onto a computer in order to detect, prevent and make decisions regarding whether to quarantine or delete malicious programs such as

malware, worms or viruses. The functionality of an anti-virus program is a running process that examines executables, worms and viruses in the memory of guarded computer/network systems instead of monitoring network traffic (Wang, Zhang and Song 2012, Bul'ajoul, James and Pannu 2013 and Al-Saleh 2015).

Although the anti-virus program monitors the integrity of data files against illegal modifications, it is unable to block unwanted network traffic intended to damage the network. Anti-threat software is installed only at explicit points of the servers such as interface between the network segment to be protected and outside environments (Shiri, Shanmugam and Idris 2011and Bul'ajoul, James and Pannu 2015).

2.3.3 Commercial state-of-the-art

Many vendors are now trying to produce security appliances that can protect networks and which combine technologies. For example, the Cisco ASA (Adaptive Security Appliance) 5500 series is a range of essential Cisco products that aims to secure an organisation's network from end to end. The main gear of Cisco Secure X design is a firewall called the adaptive security appliance (ASA). The product comes in different sizes and has been a popular choice for network designers because of its high performance. The Cisco ASA 5500 Series integrates multiple full-featured, high-performance security services, including application-aware firewall, SSL and IPsec VPN (virtual private network), IPS with Global Correlation and guaranteed coverage, antivirus, antispam, antiphishing, and web filtering services (Cisco 2016b).

The ASA series includes the essential ASA 5505 through to the ASA 5585. The differentiating feature between the ASA appliances and other software security products is that the ASA series products combine firewall, VPN concentrator and intrusion prevention in one image software. Cisco claims that this technology, which combines tools, provides a great improvement to network security. New features of the ASA technology include virtualisation, identity firewall, and threat control and containment services (Cisco 2016b).

ASA virtualization is the capability to split an ASA appliance into numerous separate devices. The feature of "high availability with failover" uses redundancy so that full availability can be provided even if an ASA devices fails. The identity firewall allows the ASA to use an organisation's active directory, which contains user and departmental policies and rules, to provide identity-based protection. Additionally the ASA appliance can be used as an IPS. The feature of "Threat control and

containment services" allows the ASA Cisco appliance to use external intrusion detection models (Cisco 2016b).

2.3.4 IDPS technology

An IDPS is a more advanced and enhanced security tool than a firewall, because a firewall just drops packets but cannot detect intrusion since the packets not examined (Whitman et al. 2012:150-164 , Weaver, Weaver, and Farwood 2013:83, Kenkre, Pai, Colaco 2015 and Yan, Jian-Wen and Lin 2015). The difference between a firewall and IDPS can be indistinguishable to the user as the separate technologies are often combined to a single gateway sentry system. The firewall checks headers on packets and blocks depending on header information such as protocol type, source address, destination address, source port, and/or destination port according to network security policy. An IDPS identifies the attacks and protects the system handling issues like misuse of the computer system, DoS/DDoS attack, and flooding attacks (Whitman et al. 2012:224, Weaver, Weaver, Farwood 2013:269, Bul'ajoul, James and Pannu 2015 and Özçelik, Brooks 2015,).

Current state-of-the-art IDPS provides highly accurate results as compared to other security protection techniques. Some researchers indicated that IDPS has significantly improved with the passage of time, but they still often produce an unacceptable quantity of false positives and false negatives (Weaver, Weaver, and Farwood 2013:83 and Amudhavel et al. 2016). In addition, it is difficult to detect suspicious activities in the midst of high traffic and other such adverse circumstances in the network, consequently resulting in an inaccurate detection mechanism. IDPS is still unable to control all threats and malicious activities (Bul'ajoul, James and Pannu 2015, Malik and Singh 2015 and Özçelik, and Brooks 2015). These issues motivated the research towards the aim of developing a solution to the under-performance of NIDPS in high-speed and high-volume traffic situations.

2.4 Intrusion Detection system (IDS) and Intrusion Prevention System (IPS)

A distinction is often made between intrusion detection systems (IDS) and intrusion prevention systems (IPS). This distinction is that the IDS detects intrusions and reports them, whereas the IPS detects reports and prevents them through blocking. Therefore the IPS can be seen as an extension of the IDS. However, nowadays the technologies have converged and most IDS systems cover prevention as well as detection. The mode of operation between detection and prevention may be selected via a configuration setting. IDS and IPS can be indistinguishable to the user as the

separate technologies are often combined to a single gateway sentry system. The IPS checks both headers and payload, blocking on recognisable known features according to network security policy. Combined Systems are often known as Intrusion Detection and Prevention Systems (IDPS) (Whitman et al. 2012:221, Weaver, Weaver, and Farwood 2013, and Bul'ajoul, James and Pannu 2015).

2.5 Types of Intrusion Detection and Prevention System (IDPS)

IDPS are often used to sniff out network packets giving the network users/administrator a clear picture of what is truly happening on the network. An IDPS also has the potential to detect and reports any evidence of attacks such as flooding attacks, unauthorised log-ins, privilege escalation, illegitimate access, modification of data and data-driven attacks. IDPS are effective and useful in controlling malicious activity and threats under circumstances where traffic is constantly growing. NIDPSs are further classified as software- or hardware-based. The mechanism of an IDPS is based on how, where and what it detects, along with mandatory requirements. In particular, IDPSs should be based on flexible and scalable network components to accommodate the drastic increase in today's network environments (Whitman et al. 2012, Bul'ajoul, James and Pannu 2013 and Bul'ajoul, James and Pannu 2015). The IDPS should provide straightforward and user-friendly management and operational procedures and steps instead of complicating its underlying tasks.

The typical actions of IDPS software can be classified as follows:

- Monitors entire and/or partial packets;
- Detects (alerts, logs and passes) or prevents (blocks, rejects and drops) suspicious activities;
- Records required events; and
- Sends updates to the network administrator.

Some of the existing types of IDPSs are: network-based (NIDPS); host-based (HIDPS); and graph-based IDS (GrIDPS). Hybrid systems also exist which combine one or more types into a single system (Whitman et al. 2012:224, Weaver, Weaver, and Farwood 2013:265-285 and Bul'ajoul, James and Pannu 2015).

2.5.1 Network-based IDPS (NIDPS)

According to Yang, Fang, Liu and Zhang (2004) and Hofstede and Pras (2012), network-based IDPS (NIDPS) have become a critical component of an organization's security solution. A NIDPS is capable of detecting a broad range of malicious and unwanted attacks occurring in the

application, network and transport layers, along with unexpected services based on multiple applications. In addition, NIDPS are able to detect and monitor the network traffic and secure the computer systems from network-based threats without network policy violations (Scarfone and Mell 2007, Whitman et al. 2012:226-230, Stanciu 2013, Arbor Networks 2015, Bul'ajoul, James and Pannu 2015, Malik and Singh 2015, Özçelik and Brooks 2015 and Li et al. 2015).

Figure 2. 3: An example of network-based IDPS (Bul'ajoul, James and Pannu, 2015).

However, an NIDPS can have disadvantages. NIDPSs are usually unable to check all incoming network packets in high-speed and high-load environments. This results in incomplete analyses and therefore considerable delays. The NIDPS itself is affected by DoS and DDoS attacks, similar to those against IPSec gateways. In addition, an NIDPS is unable to inspect encrypted network traffic (packets) due to the placement in the middle of the network connections (see Figure 2.3); similarly, eavesdroppers are unable to understand encrypted traffic in the middle of the traffic. DoS and DDoS attacks can also overcome the processing power of the NIDPS because of the attacks' speed and volume (Arbor Networks 2015 and Malik and Singh 2015). The NIDPS only works on the transporting or routing part of the network environment, as opposed to an end point of a network (see Figure 2.3). The NIDPS can be placed at the hub and can see all traffic but this is not possible in a switched network where there is no hub. In a switched network, port mirroring or spanning is used to enable a complete view but this causes overhead.

2.5.2 Host-based IDPSs

In order to overcome the problems with NIDPS discussed above, host-based IDPS (HIDPS) are implemented to monitor suspected events happening in local host machines. The HIDPS is versatile due to its installation over servers, workstations and notebooks, as compared to NIDPS. In addition, HIDPS are capable of monitoring malicious networks and multiple events happening within the protected host. A HIDPS is situated at the end point of a computer network (see Figure 2.4) same as anti-threats applications (spyware detection, firewalls and antivirus software programs), which provide the access to the outside environment such as internet. Host-based multiple IDPSs consult

various kinds of log files (e.g., system, kernel, network and server firewall) and compare logs and an internal database of common signatures for recognised attacks (Vigna and Kruegel 2006:1-13, Scarfone and Mell 2007, Topallar 2009, Whitman et al. 2012:230, Roozbhani and Rikhtechi 2010, Bul'ajoul, James and Pannu 2013, Bul'ajoul, James and Pannu 2015 and Li, et al. 2015).

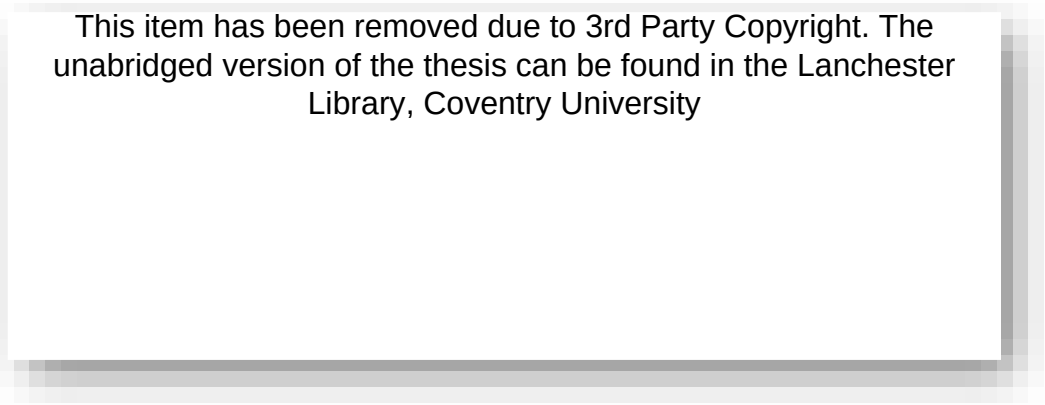


Figure 2. 4: An example of host-based IDPS.

Additional advantages of HIDPSs include:

- Capable of integrating code analysis, monitoring system calls, detecting buffer overflows, privilege misuse and abuse, file system, library list and application, system configuration and system analysis. This can be done by the HIDPS, because the HIDPS is designed to operate with a specific host and with respect to applications such as web servers, database servers, file servers, mail servers and DNS servers.
- The HIDPS is often integrated into server software and can be relatively easily implemented to communicate with other network components and operating system.
- It can inspect encrypted traffic, because the HIDPS has capability to analyse packets at the application ends.

The disadvantages of an HIDPS are as follows:

- It consumes computer system resources that should be allocated for services.
- It may conflict with existing security policies of firewalls and operating systems.
- It is difficult to analyse intrusion attempts on multiple computers.
- It can be very difficult to maintain in large networks with different operating systems and configurations.
- It can be disabled by attackers after the system is compromised.
- It requires many hosts to reboot after a complete installation or updates and many essential servers cannot support this operation (Kim, Pamnami and Patel 2007,

Scarfone and Mell 2007, Whitman et al. 2012:225-232, Bul'ajoul, James and Pannu 2013 and Bul'ajoul, James and Pannu 2015).

2.5.3 Hybrid-Based IDPS

In some situations, HIDPS and NIDPS may unable to fulfil the requirements for intrusion detection because each type of IDPS has both inherent virtues and shortcomings. Therefore, the combination of HIDPS and NIDPS is known as Hybrid IDPS (Kim, Pamnami and Patel 2007, and Li, et al. 2015). These are widely used in computers and networks for security management.

2.5.4 Graph-based IDPS (GrIDPS)

Graph-based IDPSs (GrIDPSs) are designed to protect computer networks from large-scale malicious attacks, which severely affect computer networks. Network traffic and computers are linked through GrIDPs. The advantages of GrIDSs are that they can gather data about computer activity across a network and help to recognize comprehensive automated or coordinated attacks in real time. They allow network systems to state and implement policies specifying which users are permitted to utilise the particular services of an individual host or group of hosts. Assumptions made in this kind of system include the existence of related networks within a single organisation that has an independent infrastructure and sovereign departments. It also assumes that no single component of the network is actively hostile, and therefore the IDPS must be designed to operate in non-hostile situations. (Bul'ajoul, James and Pannu 2015, Costa et al. 2015 and Fredj 2015).

2.6 Intrusion Detection and Prevention Systems (IDPS) methodology

Most IDPSs utilise either misuse detection and prevention or non-regular pattern detection and prevention. The technique of misuse detection is employed to find known intrusions and/or a pattern of signatures. Due to its reliance on signatures, its detection speed is quite fast and has a low false positive rate (Weaver, Weaver, and Farwood 2013:265-290). On the contrary, an anomaly-based method is able to detect unknown intrusions due to its intelligent detection behaviour. It is based on profiles which present the usual behavioural activities of users, systems, network connections and applications. These profiles are expanded to monitor the attributes of typical activity over a period of time (Whitman et al. 2012:233, Weaver, Weaver, and Farwood 2013:268 and Bul'ajoul, James and Pannu 2015). Profiles can be generated depending on a number of behavioural attributes such as number of emails generated by a user, the number of failed login attempts for a host, and the processor-usage level for a host in a given period of time. Defining profiles is a very important step. If

the profiles are not defined properly or are broadly defined, some attacks might not be detected, leading to a low detection rate for the computer system. On the contrary, if the profiles are too narrowly defined then various usual activities might be detected and considered as intrusion.

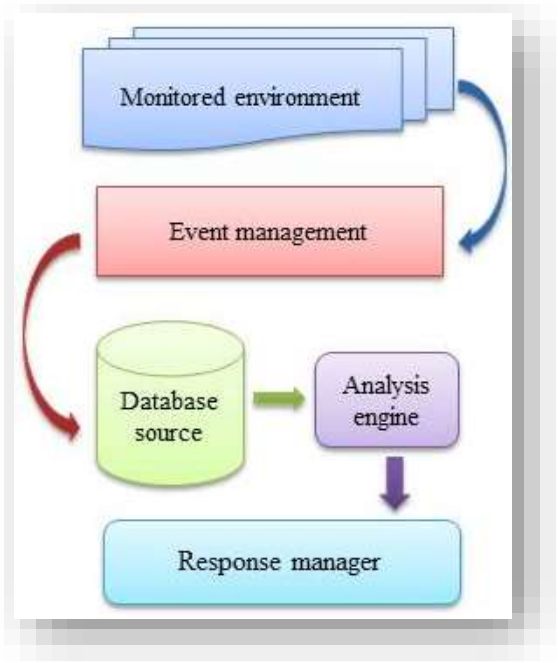


Figure 2. 5: General architecture for an IDPS.

The functional components of an integrated IDPS are: events management, data storage, analysis engine, and response manager. Event management gathers information on events (such as alerts or block events) to and from the monitored system (see Figure 2.5) and sends these to the database source. The database source stores multiple events gathered by event management. The analysis engine collects data from the data source in order to analyse and determine whether the data is free of policy violations or other attacks. This engine can utilize anomaly/statistical detection, misuse/signature-based detection, or both. The analysis engine processes events and transmits alerts. The response manager neutralizes an attack once it is detected. The response manager responds to events and stops intrusions (Whitman et al. 2012:232-254, Weaver, Mudzingwa and Agrawal 2012, Weaver, and Farwood 2013 and Bul'ajoul, James and Pannu 2015 and).

2.6.1 Signature-based IDPSs methodology

Signature detection has been used to detect known attacks. Signature-based IDPSs compare observed signatures with known attack signatures. It has a higher level of security than anomaly detection. Signature-based IDPSs cannot recognise a new attack in the monitored environment

(Mudzingwa and Agrawal 2012) and are therefore unreliable when it comes to detecting threats. The IDPS uses known signatures of malicious codes, which are stored in an IDPS database. This kind of detection system is highly efficient for use in a small IDPS. The major drawback of such a system is that its database must be regularly updated, resulting in an ever-increasing database that must include as many available signatures as possible (Hoque et al. 2012, Mudzingwa and Agrawal 2012 and Bul'ajoul, James, and Pannu 2015). Thus the checking process takes more time, which tends to weaken the performance of the IDPS.

The architecture shown in Figure 2.6 utilizes the detectors which discover and evaluate the signatures available in the monitored environment to the known signatures database. The system generates alerts if signatures match but on the contrary, the detector does not generate any alert if there is no signature match with the database (Mudzingwa and Agrawal 2012).

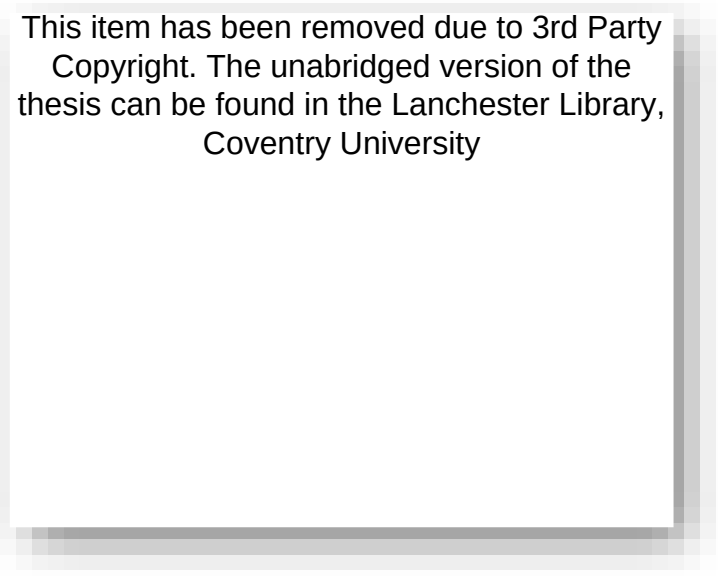


Figure 2. 6: Signature-based methodology architecture (Mudzingwa and Agrawal 2012).

2.6.2 Anomaly-based IDPS methodology

Anomaly-based IDPSs require foundation information and particular knowledge of the system being protected. Such systems have profound merit in gathering evidence in the form of statistics, data, facts and figures, which are responsible for the formation of baselines during the learning period.

The baseline profile is the normal learned behaviour of the monitored system and is developed during the learning period, while the IDPS learns the environment and develops a normal

profile of the monitored system. This environment can be a network, users or a system. Anomaly-based IDPSs are further classified as follows: Protocol-based Anomaly; and Application Payload-based Anomaly (Mudzingwa and Agrawal 2012, Whitman et al. 2012, Weaver, Weaver, and Farwood 2013:256-274 and Bul'ajoul, James and Pannu 2013). Anomaly-based IDPSs recognise breaches on computer technology and systems that are outside the normal range for standard network traffic and system operations.

Anomaly-based methodologies can identify and detect unknown intrusions and attacks on a computer network environment without requiring updates to the system. Whenever an anomalous operation is sensed, a standard anomaly-handling action should be initiated. This might occasionally lead to false positives (Chen and Chen 2009, Shiri, Shanmugam and Idris 2011 and Pal and Verma 2015).

The anomaly/statistical NIDPS method is a comparison-based method which compares any activity to the profile for all possible learned actives through statistical data, facts and figures. There are two types of profile, fixed and dynamic. A fixed profile is the most efficient as compared to other schemes, because it terminates the occurrence of any unusual behaviour and it classifies the behaviour as anomalous. A fixed profile cannot be modified once it is established, whereas a dynamic profile can be changed as the system being monitored changes. An extra overhead will be added to the system as the IDPS continues to update the dynamic profile. In the IDPS that implements a dynamic profile, an attacker can avoid detection by spreading the attack over a long time period. The attack becomes part of the profile as the IDPS incorporates the changes into the profile as normal system changes A dynamic profile cannot be created without an existing fixed profile; once the dynamic profile has been created, it allows the attacker to observe and alter his or her behaviour in long-term activities (Hoque et al. 2012, Mudzingwa and Agrawal 2012 and Bul'ajoul, James and Pannu 2015).

Anomaly detection can be used to detect new attacks, but there is no guarantee of the accuracy of the detection. It generates false positive alarms (Shiri, Shanmugam and Ideis 2011, Weaver, Weaver, and Farwood 2013:268 and Bul'ajoul, James and Pannu 2013); therefore, the problem of accuracy is still an issue for researchers (Ru et al. 2016 and Zhao, Jiang, and Stathaki 2016). According to Scarfone and Mell (2007) there are three general techniques for anomaly detection, statistical, knowledge/data-mining and machine learning.

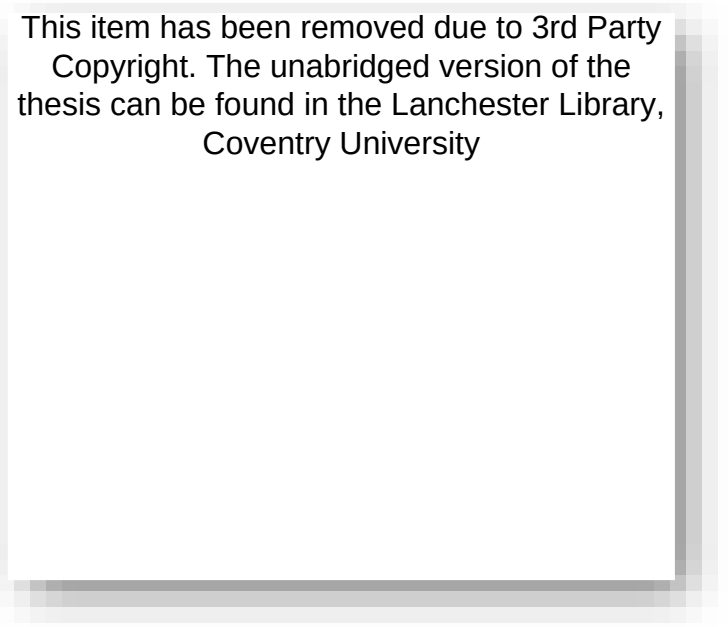


Figure 2. 7: Anomaly-based methodology architecture (Mudzingwa and Agrawal 2012).

The monitored environment is observed by the detector that matches events against a baseline profile with two possibilities. If the observed event does not equate with the baseline but lies within the range of an acceptable threshold, then the profile is updated. On the other hand, if the observed event equates with the baseline, then no action is required. If the observed event does not match the baseline profile and is outside the range of the threshold, the alert must be issued and the event marked as anomaly (see Figure 2.7) (Mudzingwa and Agrawal 2012).

2.6.3 Analysis-based stateful protocol IDPS methodology

The Stateful Protocol IDPS methodology incorporates the notion of state and is therefore capable of understanding and tracking the network protocol state. Stateful protocol models are built on TCP/IP protocols using their specifications. The stateful protocol analysis models are built on TCP/IP protocols using their specifications. The stateful protocol NIDPS technique is based on analysis of the behaviour of the protocols. It observes the protocol behaviour and then compares it to those stored in its protocol behaviour database. It detects anomalies in the packet on the head part of the protocol. This technique is quite effective, but can be easily avoided by attackers working inside the protocol limitations (Hoque et al. 2012 and Bul'ajoul, James and Pannu 2015).

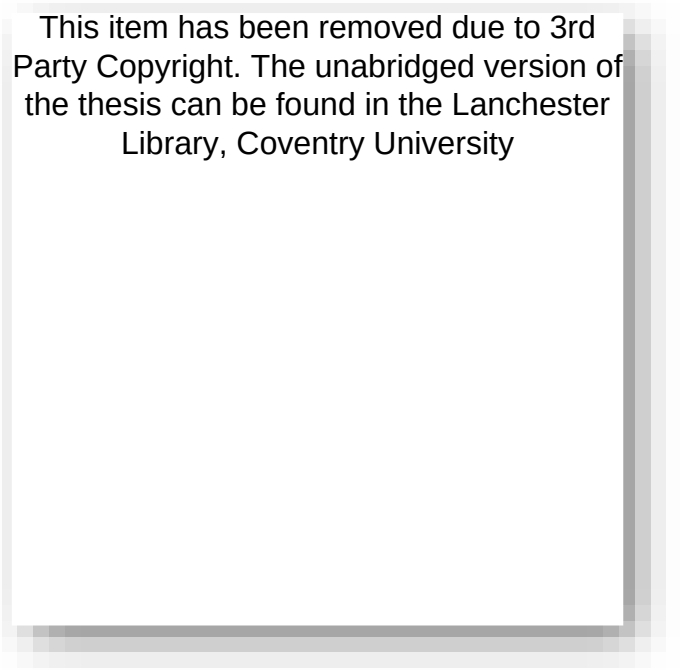


Figure 2. 8: Analysis-based stateful protocol architecture (Mudzingwa and Agrawal 2012).

Multiple vendors established and designed a baseline profile of the protocols. Stateful protocol analysis provides in-depth understanding of related applications and protocols and how they interact and work each other, but it introduces additional overheads in the system (Scarfone and Mell 2007 and Mudzingwa and Agrawal 2012).

Furthermore, intruders usually use signatures which behave similarly to viruses used in computers. The protocol anomaly detection method analyses data packets related to intrusion, which contain known anomalies and single or sets of signatures. The detection system is capable of detecting suspicious activity in the logs and generates alerts based on these signatures and rules. Anomaly-based IDSs generally depend on detecting packet anomalies available in the header parts of the protocol. The universal architecture of the Stateful protocol and its analysis is similar to the methodology of the signature-based approach (see Figure 2.8) and requires a database of acceptable protocol behaviours.

2.6.4 Hybrid IDPS methodologies

The hybrid system is the integration of two or more methods. Hybrid methodology can combine two or more intrusion detection and prevention systems methodologies in order to analyse, detect and match any suspicious behaviour and signature-based malicious code that attempt to attack the network. The power of combination means it can detect more types of intrusion, thereby providing

relatively better results as compared to other methods. Figure 2.9 depicts the behaviour of the general hybrid methodology which combines stateful protocol analysis, signature and anomaly methods. The monitored environment is analysed by each method in turn (Mudzingwa and Agrawal 2012and Bul'ajoul, James and Pannu 2015).

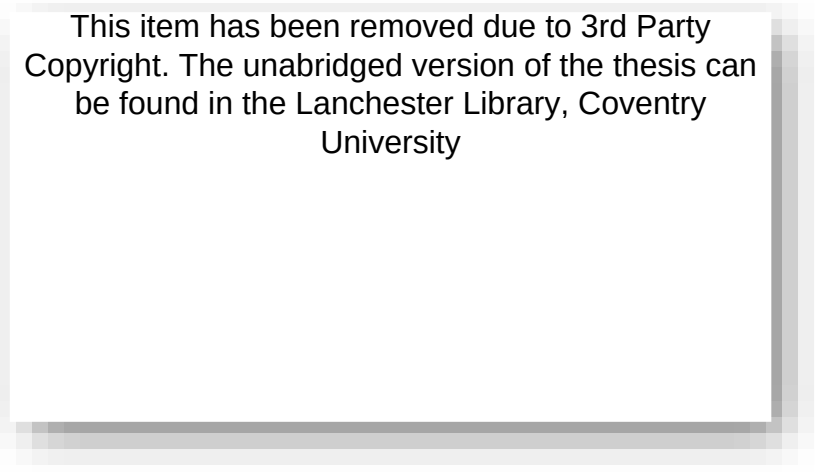


Figure 2. 9: Hybrid methodology architecture (Mudzingwa and Agrawal 2012).

2.7 Open Source NIDPSs (Snort and Bro)

Snort (2016) and Bro (2014) are both well-known open-source IDPSs. This research uses the Snort open-source IDPS. In this section some comments are made on these two main open-source IDPSs and reasons are given on why Snort was adopted as the vehicle to demonstrate the novel architecture developed in this research.

Snort is ranked among the top NIDPSs currently available. In spite of the huge development over the years and offering a de facto open-source IDP/IPS solution for many years (Khalil 2015), Snort is still struggling to sustain its growth in the network industry and prevent attacks (Bul'ajoul, James and Pannu 2015, Podofillini et al. 2015 and Wang and Kissel 2015). It was released as an open-source, rule-centred NIDPS, which stores rules in text files that can be modified by a text editor. The rules are grouped into categories, and the rules belonging to each category are stored as information in separate files; these files are then integrated into the main configuration file, named "snort.conf". The data is captured in terms based on described rules, which are read at the initialisation of Snort and are used to construct the internal data structure.

Although, Snort is one of the most useable and popular NIDPS tools, other NIDPSs such as Bro, have received attention from the researchers (Gupta 2012 and Khalil 2015). Bro is a UNIX

based, open-source NIDPS capable of monitoring network traffic using passive methods to observe suspicious and malicious activity. It detects specific attacks based on event signatures. Some studies (Khalil 2015, Jaiswal, Lokhande and Gulavani 2015 and Stocks 2015) compare Snort and Bro are on the basis of various parameters such as performance, processing speed, signatures, flexibility, deployment, interface and capability of operating system.

Bro is getting popular in the research area due to its flexibility and easy customization. It is also useful for the advanced techniques of detection and is easy to integrate with existing tools. Association and correlation are other important and useful features which secure and protect the overall network system. The performance of Bro is higher than Snort in terms of correlation of events. Bro users use the sys-log as an output which provides the broad results of the events in the system. These features are not available in Snort but researchers are working to add such features to Snort (Mehra 2012, Jaiswal, Lokhande and Gulavani 2015 and Stocks 2015). A multi-instance feature has been added to Snort whereby a new Snort instance can be launched across each core to support load balancing across multiple CPUs. Snort has become multi-threading to address high-throughput networks. However, Bro lacks built-in, multi-threading.

Bro and Snort have the ability handle high-speed traffic. This makes them suitable for larger scale Gbps networks but both have limits. Both are flexible and can be configured for their intended computer network. Bro includes pre-written rule scripts which cannot be modified and these will detect the most well-known attacks. More features can be added and policy scripts can be customised to cope with new attacks. The policy scripts can be customised to contain application-specific rules. Snort rules are powerful, flexible and relatively easy to write and also have provision for customization.

Bro differs from Snort in its event-driven analysis. It has its own policy engine, which provides the capabilities of analysis, downloading of files on the wire, and then notifying the administrator. When a computer user tries to modify or download its script, the event engine will stop and shutdown. Snort does not have its own facility to capture packets. An external packets-sniffing library is required. Often used is the LipPcap library, which is widely supported across various operating systems. When Libpacp packets are delivered to Snort, the packets will be processed through a series of decoders corresponding to the protocol stack elements. Once the packets are decoded, they move up to the pre-processor and detection engine for analysis.

Arguably Snort is easier to use than Bro because it has a graphical interface Therefore it more popular. Snort can run on most of today’s common OSs such as Windows, UNIX, Mac, and Linux including virtual machines whereas Bro is limited to UNIX based OSs. Snort can be configured to

different IDS/IPS modes such as Sniffing, passive and inline mode. Bro does not support inline intrusion prevention (IP) mode; it offers a script-driven IDS.

Bro architecture is different from Snort. It is a script-driven policy engine rather than processing and decoding engine. Snort offers a wide variety of pre-processors which examine and modify packets for input to the detection engine which has rules for parsing and signature detection. New functionality can be added in Bro through the creation of policy scripts which can be written in the Bro. New functionality in Snort is written in the C language (Mehra 2012, Stocks 2015, Jaiswal, Lokhande, and Gulavani 2015 and Khalil 2015).

Several NIDPSs are available on marketplace and some open-source NIDPSs are also available. Snort and Bro are free-of-charge NIDPSs and both are available for download from their respective websites. Snort, Bro and other freeware NIDPS systems are often used both in research and operational systems (Bro 2014, Snort 2016). According to the Snort website (Snort 2016), the Snort community includes more than 500,000 registered and active users. Also, there have been over four (4) millions downloads from national and international universities which are actively using Snort for their research purposes and tutorials. Snort is the most widely deployed IDPS in the world. On the contrary, there are no numbers published in the community to show the users of Bro and it is assumed that the user base is smaller. Snort has been developed to be suitable for speedy networks and is packet-oriented whereas Bro is connection-oriented (Mehra 2012, Chen et al. 2015, Kenkre, Pai and Colaco 2015 and Bul'ajoul, James and Pannu 2015). Snort focuses on performance and simplicity and is one of the best known lightweight NIDPSs.

It was decided to use Snort instead of Bro in this research because of its larger user base, longer established research community and ability to be used on multiple operating systems.

2.8 Use of Parallelism in Intrusion Detection

Multi-core technology and parallelism is a possible solution for the high-speed network traffic security problem. This section describes relevant related work in parallelism in intrusion detection.

Salah and Qahtan (2009) implemented a hybrid scheme in Linux OS to prove that a hybrid scheme can improve the performance of general-purpose network desktops or servers running network I/O-band applications when such network hosts were exposed to both light and heavy traffic load conditions. In order to achieve a high throughput of analysed traffic, the researchers tuned the budget parameter of the Linux Network subsystem. This parameter controls the utilization time of the

central processing unit cycle. However, this solution will cause another problem, which is polling of speedy and large traffic to CPU for which there is no buffer. If the incoming traffic speed is higher than CPU processing speed, some of the traffic will be dropped. The method proposed in this thesis is similar to the work of Salah and Qahtan in that it exploits configuration in a general purpose environment, but it is different in that it introduces a novel configuration of QoS at a multi-layer switch level by using a buffer reservation technique together with parallel processing technology. Thus it is network-based rather than the host-based.

Shiri, Shanmugam and Idris (2011) proposed a type of parallelism to improve the performance of signature based IDSs. They used two signatures and created parallel implementations of Snort. Then they distributed the traffic between the two Snort nodes, each node handling one of the signatures. Beg et al. (2010) gave an overview about using AI (Artificial Intelligence) techniques in IDS and showed their capabilities for intrusion handling and minimising false alarms. They provided details about how using different AI approaches in IDS has serious disadvantages in a large and high-speed network. The variations in the AI algorithms make it quite difficult to pin-point exact limitations of the system or the technique used. Moreover, for a large, high-speed network, corresponding computation needs arise which make AI algorithms more difficult and less scalable. Beg et al. suggested the use of HPC in a centralized network to improve processing of large data and speedy traffic in the context of using AI techniques. This thesis has similarities to the research of Beg et al. in that it uses multiple Snort nodes and parallelism but it addresses performance in a different way. It explores the use standard network components to improve processing of a speedy and heavy traffic through improved QoS architecture.

One of the most important weaknesses of NIDPSs is that they fail, when processing unexpected increases in traffic volume. It is crucial that more efficient approaches are developed. To find a solution to this problem, several nodes can be used to process network traffic concurrently and in parallel. Wheeler and Fulp (2007) proposed a framework that is complementary to NIDPS. Their research illustrates that three levels of parallelism can be used:

- the node level (the node plays a vital role in the running of several identical systems (entire systems) in a parallel fashion: the same set of tasks (or rules) are replicated at each node);
- the component level (defines a form of practical parallelism; a fraction of tasks (rules) are set for each node and given their own processing elements); and

- the sub-component level (refers to the further parallelisation of individual components). The parallelisation of pre-processing into critical and non-critical pre-processing can be viewed as a functional sub-component parallelism).

In Snort, rules are placed into rule groups based on their source and destination: for example, rules associated with web traffic are usually placed in port number 80. In the work of Wheeler and Fulp, Snort organises the rules into discrete groups, and each individual group is commonly recognised by its file name. Packet duplicators are used to duplicate incoming packets that run across all the nodes at the same time, because of different tasks (or rules) maybe maintained at different nodes. In the node level parallelism, one can therefore assume that one packet may pass through the same inspection many times. The main flaw of this method is repetition, so that when a packet is sent to the node, the node must check whether its rules are related to that of the packet. Wheeler and Fulp (2007) do not cover this issue in their research.

There are many difficulties associated with node level parallelism:

- If all communications are considered to be stateless, the node level is able to work. However, this seems to be unrealistic with today's attack, as demonstrated by (Hernandez-Herrero and Solworth 2007 and Shaikh et al. 2009);
- Duplication of processing may occur if a packet is sent to more than one node. In contrast, in the method proposed in this thesis, traffic (packets) was configured and treated by using QoS configuration and other switch technology such as ACL, Queues, bandwidth, threshold and DiffServ architecture to help prevent duplications of packet processing at multiple NIDPS nodes.
- No co-relation was reflected in the packets that were sent in multiple frames; in contrast, the method proposed in this thesis classifies and processes the traffic through specific class and policy maps and then packets are passed to multiple NIDPS nodes, which analyse traffic depending on packet IP frames.
- A flaw in the node level parallelism method further reveals that it does not take into account the fragmentation issue, which is the latest primary technique used by abusers who overflow systems. The latest IDPSs are unable to handle the process of fragmentation because of the sheer speed of the attacks (Vasanthi and Chandrasekar 2011). In the novel architecture configuration proposed in this thesis, the traffic speed has been organised and controlled wherever the speed is high to help prevent the effectiveness of increasing attack speed. QoS queuing, DSCP (Differentiated Service Code Protocol) and SRR (Share and Shape Round Robin) were used on lower

bandwidth to ensure real-time network traffic does not suffer from high jitter and delay.

Wheeler and Fulp (2007) demonstrate that at the component level, some particular functions, including e-fragmentation, can be parallelised. However, it has not been properly clarified how this will take place in this system, and the risk may be exacerbated by the formation of a bottleneck at this level. This can even be increased if the top-level categorisation is not done in order to isolate the fragmented packets from whole and complete packets.

Shiri, Shanmugam and Idris (2011) proposed a parallel technique for improving the performance of a signature-based NIDPS. Their idea was to send different types of packets to different parallel Snorts for analysis and they obtained a 40% improvement in processing time. Schuff, Choe and Pai (2007) proposed a multi-thread Snort called MultiSnort which executes multiple instances of the original Snort in parallel. The research of this thesis is similar to these in that different types of packet are sent to parallel Snorts. However, while the research of this thesis confirms the findings of previous research, the main difference is that it provides new detail on how to achieve the improvement through QoS and parallelisation using industry standard software systems. Another difference is that it has concentrated on NIDPS analysis, detection and prevention and also provided further experiments with greater detail of relevant parameters.

Chen et al. (2009) presented Para-Snort which revised the structure of the original Snort decoupling the decoding part so that this activity is carried out centrally before the parallel queues are formed. The approach also used central load balancing to distribute packets to parallel Snort processing units. The research of this thesis differs from Chen et al.'s work in that the whole of Snort is parallelised (including decoding). Parallel queues are formed in the switch before being sent to Snort pre-processing and decoding. The problem with central decoding, pre-processing and load balancing modules is that they could become additional bottle necks in the system. Chen et al. also researched how to reduce the load balancing bottle neck issue.

Vasiliadis, Polychronakis, and Ioannidis (2011) proposed a new model for a multi-parallel IDS architecture (MIDeA) for high-performance processing and stateful analysis of network traffic. Their solution offers parallelism at a subcomponent level, with NICs, CPUs and GPUs doing specialised tasks to improve scalability and running time. They showed that processing speeds can reach up to 5.2Gbps with zero packet loss in a multi-processor system. Jiang et al. (2013) proposed a parallel design for NIDS on a TILERAGX36 many-core processor. They explored data and pipeline parallelism and optimized the architecture by exploiting existing features of TILERAGX36 to break the bottlenecks in the parallel design. They designed a system for parallel network traffic processing

for implementing an NIDS on the TILERAGX36 which has a 36 core processor (Tilera). The system was designed according to two strategies: first a hybrid parallel architecture was used, combining data and pipeline parallelism; and secondly a hybrid load-balancing scheme was used. They took advantage of the parallelism offered by combining data, pipeline parallelism and multiple cores, using both rule-set and flow space partitioning. They showed that processing speeds can handle and reach up to 7.2Gbpswith 100-bytes packets and 13.5 Gbps for 512-bytes. Jamshed et al. (2012) presented Kargus's system which exploits high processing parallelism by balancing the pattern matching workloads with multi-core CPUs and heterogeneous GPUs. Kargus adapts its resource usage depending on the input rate, to save power. The research shows that Kargus handles up to 33 Gbps of normal traffic and achieves 9 to 10 Gbps even when all packets contain attack signatures. The various approaches described in this paragraph are not directly comparable in terms of throughput as different numbers of processors is used in each. However the experiments show that high gains can be made by parallelising NIDPSs in order to combat problems of higher speeds and increasing traffic. The novel research presented in this thesis differs from the research described in this paragraph in that the research presented in this thesis shows how QoS and queue technologies can be exploited in a multi-layer switch to improve packets processing performance. Further enhancements occur when queuing is combined with parallel processor technologies. The other research did not exploit QoS. The approach of this thesis has shown how parallelism at a lower level of granularity, which is simpler to implement, can also make impressive improvements for NIDPS performance in terms of throughput and the number of dropped packages.

Chen, et al. (2015) proposed an application-specific integrated circuit (ASIC) design with parallel exact matching (PEM) architecture to accelerate processor packets speed. The ASIC hardware has been designed to operate at 435MHz to perform up to 13.9 Gbps throughput to manage the requirements of high-speed and high accuracy for IDS, which resolves the issue of the information security limitation to manage data received from the 10Gbps core network. They proposed SRA (Snort Rule Accelerator) with parallel rules to increase the performance of the IDS. The SRA is proposed with a stateless parallel-matching scheme to perform high throughput packet filtering as an accelerator of the Snort detection engine. The ASIC is composed of five major modules, including the Inspector, Counter, Parallel Matching, Conformity and Compare modules. The parallel matching scheme compares a packet's payload with the stored rule. When an entry packet is matched with Snort rules, the ASIC is in an idle state and sends a compare signal to the conformity module, which integrates all signals and determines whether an abnormal payload is presented. Here the authors designed a half mesh architecture in the parallel matching rules module, which allows the traffic to be compared with several rules. The research of this thesis addressed performance in a different way. It utilised hardware Layer-3 switch technology (QoS, memory and buffer dynamic reservation and parallel queue technologies) to improve network forward-throughput-traffic architecture and, hence,

NIDPS performance. It configured an interface into queues (interface-to-queues), which allows packets to be processed through the component level parallel NIDPS nodes. The approach is designed to deal with the limitation of real networks speed and finds solutions to the problems that caused the NIDPS performance. The approach can deal with any incoming traffic-speed that may allow malicious packets to enter the system and prevent NIDPS from detecting or preventing them. It does this by imposing advanced management of network packets traffic. The advantage of the proposed approach is that every-day equipment can be utilised in a new way to achieve improvements and it is also more scalable than the proposal of Chen et al. (2015).

In the context of big data and distributed systems, Zhao et al. (2014) have developed a security framework in G-Hadoop. This work focuses on authentication and access rather than intrusion detection but offers an interesting new direction. The framework could be enhanced with intrusion detection and protection functionality to create a more complete solution. The research of this thesis has focussed on standard business infrastructure whereas the work of Zhao et al. has concentrated on single cluster across cloud data centres. Cross-cluster security services in a high performance environment such as that afforded by G-Hadoop is an area where attention is welcome.

Vendor companies are aiming to develop security solutions to protect the enterprise network. Equipment has been designed to meet connectivity speed and load standards. The improvements in the throughput of NIDPS shown in this thesis are achieved by pairing the ASA Cisco equipment (Cisco 2016b) with multiple implementations of Snort. The principles of the method proposed in this thesis could be applied to other equipment combinations where similar facilities are offered.

Interesting work is being carried out in the classification of internet traffic which can be used to support attack detection. In order to counteract limitations of current internet traffic classification techniques, which are based only on header and payload inspection, Wang et al. (2014) have developed a system which can classify traffic in terms of their intended application by considering packet and flow characteristics. A machine learning approach has been used to develop the classifier. Extra complexity introduced by more demanding methods such as machine learning, albeit with the purpose of producing better detection performance, supports the contention of this thesis for a solution based on parallel NIDPS in high-speed and heavy traffic environments.

To summarise, the research of this thesis differs from the research described in this section in terms of the architecture used. The research of this thesis investigates how QoS technology and parallelism can have impact in high-speed and heavy traffic network using an industry standard switch and standard desktop processors. This solution is a more accessible way of receiving good results as it can be activated at a higher level, namely at the level of configuring the CISCO switch

software and replicating Snort on standard machines. Further improvements could be made if higher performance equipment was used. However, there is room for various approaches in the security arena and more exploration of the suitability of various methods in varying circumstances is welcome. Also the cost is generally an important concern. The design proposed in this research benefits the network security requirements at low cost.

2.9 Conclusion

In this chapter an overview has been given of the ever-growing problem of network and system attacks and the current technology used to combat them, including its limitations. It has also discussed related research work which, like this research, aims to improve the performance of NIDPSs. The chapter has also described how the research presented in this thesis differs from related research.

CHAPTER 3: METHODOLOGY AND EXPERIMENTAL DESIGN

3.1 Introduction

This chapter describes the methodology and experimental design of this research. First the chapter provides a description of the overall approach applied in this research. A major part of the approach included practical experiments which were set up using a number of technologies including Snort as the base NIDPS. The proposed solution to the research problem involved the use of QoS configuration in a Layer 3 Cisco switch together with parallel NIDS technology. The experimental testbed also incorporated generator traffic tools, such as NetScanPro, Packets Generator, Packets Traceroute, TCP reply and Packets flooder. Thus after describing the general approach to the research, relevant aspects of the participating technologies in the experimentation and solution are described. The NIDPS methodology used was the signature-based and thus the chapter also includes coverage of this. Finally, the chapter informs readers about the actual experiments carried out and the performance metrics used.

3.2 General Approach to the Research

The general method used in this research was quantitative including experimentation, inferential and simulation techniques (see section 1.5). The research followed the steps given below (also see Figure 3.1).

1. Literature Review

   A literature review was undertaken to establish the state of the art and clearly define the problem to be solved.

2. Research design

   A method to carry out the research was designed. An experimental approach was determined. It was decided to first analyse the problem through a set of practical experiments (stage 1), further tested with different tasks carried out with some additional virtual experiments (stage 2). Finally, design a solution to the problem and evaluate the solution through a second set of experiments (stage 3).

3. Analysis of the Problem (Stage 1 and Stage 2)

   The problem was exposed through a set of experiments, the results of which were analysed. The experiments show the rates of dropped packets when a network is subjected to high-speed and high-volume traffic. The first sets of experiments were carried out in a specially designed testbed. The second set of experiments was carried out in a virtual environment to investigate the reason of problem. The experiments show how the rates of dropped packets are different under different OSs, buffer size and processor speed.

4. Solution Design and Evaluation (Stage 3)

   A solution to the problem based on Layer 3 switch configuration, QoS and parallel technology was designed and implemented. The solution was evaluated through a third set of experiments.

Figure 3. 1: Main steps of research.

3.3 The NIDPS used - Snort NIDPS

Snort, a software-based NIDPS, was used as the example NIDPS for the research. Snort is one of the strongest and most popular open-source NIDPSs. Its architecture is represented in Figure 3.2. When a packet arrives at the network, Snort listens and captures packets. In the beginning, the packet decoder receives packets from multiple network interfaces, such as Point-to-Point Protocol (PPP) or Ethernet and Serial-Line-Internet-Protocol (SLIP), and then pre-processes such packets ready for the detection engine (filters organise and modify the data packets before transferring them to a detection engine). The detection engine performs three main tasks: sniffing (analysis), detection and prevention. It can perform network traffic analysis and content searching/matching in both real-time and for forensic post-processing (Caswell and Beale 2004:170, Chi 2014 and Snort 2016). Snort can be configured in three main modes: sniffer, packet logger, and network intrusion detection and prevention mode (NIDP mode).

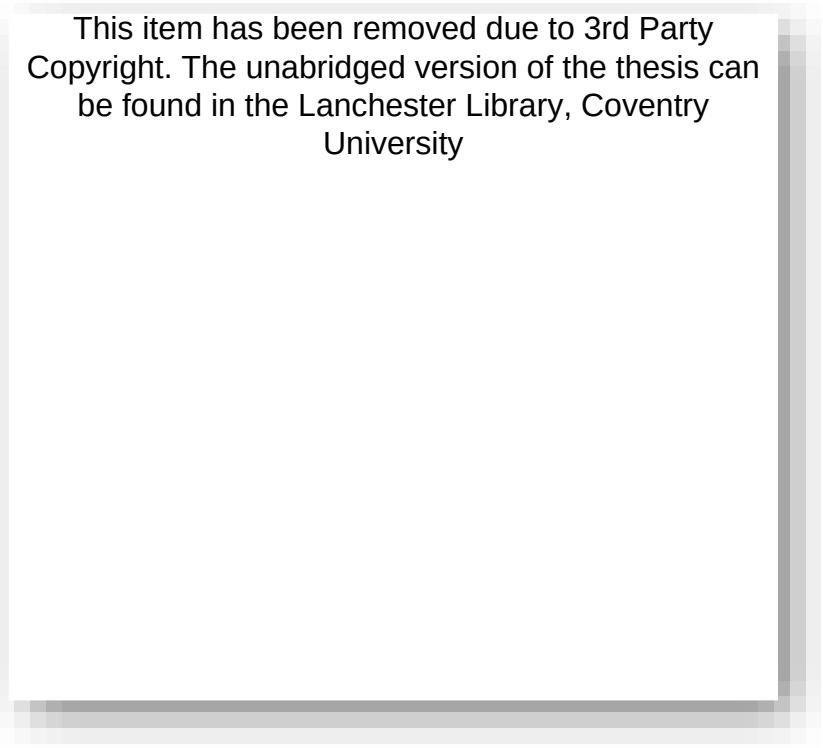


Figure 3. 2: Snort architecture (Bul'ajoul, James and Pannu 2013).

In NIDP mode, Snort analyses the network traffic against a set of defined rules in order to detect/prevent intrusion threats. In the experiments, the researcher focussed on the Snort capabilities in network intrusion detection and prevention in order to determine how many packets could be analysed, detected and prevented by Snort under varying conditions. It has been shown previously that in high-speed and heavy load conditions, packets are dropped and left outstanding and therefore

not processed properly (Balkanli 2015, Bul'ajoul, James and Pannu 2015 and Dhakar and Tiwari 2015).

The detection engine is time-critical and the most important part of the Snort. It utilises different processing times based on the length of the packet, the specifications of the system and the number of rules defined in the system. Snort sometimes drops or leaves outstanding packets when it runs in real-time NIDPS mode, particularly when traffic is heavy and high-volume Snort rules are employed to detect intrusive actions in the data packet. In NIDPS mode, Snort is capable of reading chains (internal data structures), which have to be matched against all packets. If a packet does not match any rule, it will be blocked; otherwise, appropriate action is taken.

Logs and alerts depend on the nature of what is detected inside the packets. If any suspicious activity is found inside a packet, the packet usually logs the malicious activity and/or generates an alert. Logs are usually stored in simple text-based files, such as tcp-dump files. Output modules (plug-ins) are capable of performing multiple operations depending on the results generated by the Snort log and alert system. In general, output modules control the form of outcome produced by the log and alert system.

Snort is a combination of both basic signature code analysis and content-driven rules. Snort can execute a protocol analysis and a search and match of the content. It can be utilised for the detection of various attacks and probes, such as those regarding stealth port scans, buffer overflow, SMB (Server Message Block) probes, CGI (Common Gateway Interfaces) attacks, fingerprinting attempts of OS and many more. Snort uses the rules, provided by developers or security analysts, to identify the traffic types that can be passed or collected (Bul'ajoul, James and Pannu 2015, Edge and O'Donnell 2016 and Snort 2016).

There are several features available for Snort; the most common feature is its real-time alert mechanism. Alerts can also be collected by using a mechanism called syslog, which allows the reporting of suspicious activities in various ways: logs for additional investigation; a UNIX socket;a specified file of the user; or a Win-Popup message to the window client (Snort 2016). Snort is different from other packet sniffers, due to the tcp-dump sniffer, which can be run by different operating systems and the use of the hex-dump payload dump that the tcp-dump has employed during recent years. Snort also has the capability to display packets in different networks through the same method. The default detection method of Snort in NIDPS is the signature method. The alerts are stored and activated to any system log, database, management team or a trap.

3.4 Snort Rules

Snort rules activate on an TCP/IP network and protocol layer. The part containing options normally also covers an alert message and information regarding relevant parts of the packets that can be used to generate an alert message. The options area keeps additional matching criteria. The rule can detect or prevent one or more types of interruption activity. Good rules should cover multiple intrusion signatures. Snort's rules consist of two logical parts: a rule header; and rule options (See Figure 3.3) (Weaver, Weaver and Farwood 2013:284-304 and Snort 2016).

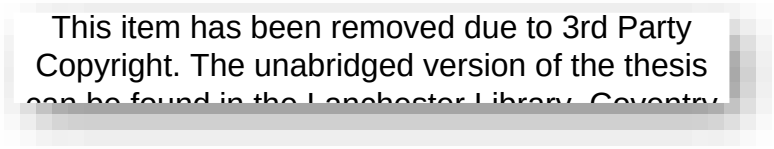


Figure 3. 3: Basic structure of Snort rules (Rehman 2003:79).

The rule header keeps the information regarding action taken by the rule and stores the criteria for matching a rule against data packets. A Snort rule structure can be seen in Figure 3.4.

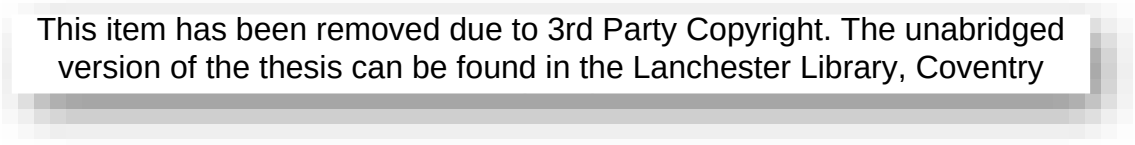


Figure 3. 4: Structure of Snort rule header (Rehman 2003:79).

3.4.1 Rule Header

The rule header consists of several different parts:

Action: The action rule is used to perform a typical function to generate an alert or log message or to initiate another rule. This performance is the first part of the rule, which shows the predicted action that will be taken when rule conditions are met. The action will take place in the case that all conditions mentioned in the rule are true.

Protocol: The application of the protocol is used only for a particular protocol. This is the second mentioned criteria of a rule. It shows the protocol to which the rule applies

(e.g. IP, UDP, TCP and ICMP). If a protocol is TCP or UDP, then Snort checks the transport layer to determine the packet's type.

Address: The address is utilised to define source and destination addresses on a rule. Addresses may consist of one or numerous IP hosts of network addresses. The address fields in rules are of two types: a destination address and a source address. Both addresses can be determined on the basis of the direction field. For example, if the direction field is '->', the source address is on left side and the destination address is on right side of a given address. For example, the following rule is to generate an alert message whenever it detects any ICMP packet from source address 10.0.0.0/8, any source port to any destination address and any destination port with time to live (ttl) of ICMP packets equal to 138:

Alert icmp[10.0.0.0/8] any -> any any (msg: "ping with ttl=138 "; ttl: 138; sid: 100001 ;).

Port: The port number is employed to apply a rule to a packet, which originated from or travelled to a specific interface (port) or range of interfaces. These port numbers are very useful for applying a rule to a specific type of traffic (packets). For example, if the point of network security weaknesses is related only to a HTTP (Hyper Text Transfer Protocol) web server, port 80 can be used in a rule to detect any attempts at exploitation. Moreover, port range, lower and upper boundaries, and the symbol of negation can be used to write an effective rule. The sample rule below would detect any UDP packets that come from any source address and specific source port range (1024 –2044) to any destination address and ports:

Alert udp any 1024:2044 -> any any (msg:" udp ports"; sid: 100002 ;).

The parts of the port identify the source and destination ports of the packets where the rules are applied. Due to the source address and port address, the rule can be applied to packets that are coming from a source port, which is related to the type of packet. If the destination IP and port are set to "any", the rule will apply to all IP address traffic (packets), irrespective of their destination address. Port numbers will be related if the protocol is either TCP or UDP.

Direction: The direction identifies which address and port numbers are used as a source and a destination.

Rule options follow the rule header and appear within the pair of parentheses. It is possible that there may be single or multiple options, which may be separated by a semicolon. The rule header is checked when relevant true criteria are found in options. Keywords are used to define all rule options, although some rule options also include arguments. A rule header can also define other rule header action, such as dropping, blocking or rejecting a rule.

There are five predefined actions that comprise the rule header: Pass, Log, Alert, Activate and Dynamic and User Defined Actions (Snort 2016). The Alert and Log are the two most common actions, which were used in the experiments and also some user-defined actions, such as Drop, Prevent and Block. The Alert action rule sends an alert message in case true rule conditions are met for particular packets. There are multiple ways to send alerts. It can be sent to a file or console. In the experiments, alerts were sent to a text file. The Log action uses to record packets details, and different methods are used to accomplish this. For example, a packets detail logs to a database or a file. Packets and headers can be recorded with several levels of detail, depending on the configuration file and command line arguments. Log and Alert have different functions: the Alert action performs a task to send an alert message and then record (log) the packet, whereas the only task of the Log action is to log the packet.

The Pass action rule is a process to inform Snort to ignore the packets. The Activate action generates an alert and then to activate another rule in order to check further conditions. The Activate rule is utilised if further testing is required for a captured packet. The Dynamic action is initiated by another rule, by using the Activate action. In fact, it can be activated only by using the Activate action, which already defines in another rule.

The User Defined action is rule actions devised by the user. These rule actions are employed for various purposes, such as sending messages to syslog, taking multiple actions on packets, and logging messages into a database such as MySQL the preventative action rules, (e.g. Block, Reject or Drop) are implemented to prevent the unwanted traffic from entering the system.

3.4.2 Rule Options

Generally, an option in a rule can consist of two parts: an argument and a keyword (Snort 2016). Colons are used to separate arguments from keywords of the option. There are various kinds of optional keyword rules, like ttl, content, offset, content list, dsize, depth and msg. (Snort 2016).

The ttl keyword rule checks the IP time-to-live values. This rule has the ability to prevent any packet reassembly. The ttl keyword can be used for many protocols constructed on the TCP/IP protocols including ICMP and UDP headers. The content rule detects a pattern in the packet. It enables the user to look for a specific content within the packet payload and activate a response. The offset keyword enables the means to start the search for a pattern from a particular point within a packet. The depth keyword enables specification of how far along the packet the search should continue. The criteria for a Snort packet search for a specific pattern depth, which modifies the previous content keyword in the rule. The offset and depth options are used to specify where to start searching (offset) for a particular content in the payload and where to stop (depth).The dsize keyword looks for payloads of the specified size (e.g. >800). It checks for abnormalities in the size of the packets, which can become a cause of buffer overflow. The content list keyword is generally used with a file name as an argument. The file contains a list of strings to be sought inside the packet. Every string is placed on a separate line of the file. The msg keyword is used in rule options to add a text string for use in logs and alerts (Snort 2016).

3.5 The layer 3 Cisco Catalyst switch technology

A layer 3 Cisco switch was used in the proposed solution. A layer 3 Cisco switch is a high-performance switch optimised for the LAN/WLAN or internet, providing wire-speed switch interface services. The switch performs three major purposes: packet switching, packet route processing, and intelligent network services. Layer 3 Cisco switches improve network performance through a variety of means: Quality of Service (QoS); DiffServ, which is based on DSCP (Differentiated Services Code Point) or IP precedence values; packets classification and modification services; rating limiting; ACLs (assess control lists); high-performance IP routing while maintain the simplicity of traditional LAN switching; and route processing queues.

A layer 3 switch contains a group of routing protocols based on Cisco IOS software, network protocols, such as IP and IPX, and routing protocols such as RIP (Router Information Protocol), OSPF (Open Shortest Path First), IGRP (Interior Gateway Routing Protocol). (Szigeti et al. 2013 and Cisco 2016a). The switch offers granular QoS features which help ensure that network traffic is classified and prioritised, and that congestion is avoided in the best possible manner.

QoS technology is identified to provide a different treatment for network traffic in different classes, which can be assigned to a specific QoS. The class to which the packet belongs can determine or discard the packet's scheduling and policies. Implementing QoS including technologies, such as Ingress and Egress Queues (IEQ), and Shaped or Share Round Robin (SRR), can help to control

traffic bandwidth, network traffic delay and packet drop. QoS technology supports some features including local administrative policy and DiffServ architecture, which can deal with different TCP/IP traffic (Cisco 2014a and Cisco2016a).

Furthermore, Layer 3 switches offer a range of security features such as enabling businesses to protect important information, keep unauthorized traffic off the network, guard privacy, and maintain uninterrupted operation. The switch can deal with malicious traffic and high-speed attacks such as flooding, malicious traffic, and DoS by using ACL (Access Control List) services, which can restrict access to sensitive portions of the network by denying packets based on source and destination MAC/IP addresses, or TCP/UDP ports. The switch delivers high-performance IP routing architecture. This architecture offers a speedy lookups while helping ensure the stability and scalability necessary to meet the needs of experiment requirements. The switches support extra features, such as "constrained multicast flooding (CMF), IP routing, IP multicast routing, routing protocol convergence with Routing Information Protocol (RIP), EtherChannel and load sharing across equal cost Layer 3 paths and spanning trees (for Layer 2 based networks)" (Cisco 2014a and Cisco2016a). They also support remote monitoring (RMON) groups. RMON (Remote Network MONitoring) is a network managing protocol for gathering network information and checking traffic flow data within remote LAN segments. RMON lets permitted users to see all traffic nodes and their interaction on a LAN segment. In the router, RMON allows the configurable user to view traffic that flows through the router by conjunction with the SNMP (Simple Network Management Protocol) agent. A Layer 3 switch combines RMON events and actions with existing MIBs (Management Information Bases) so the configurable user can choose where monitoring will occur (Cisco 2016a and Cisco 2014a).

A load balancing function distributes packets (traffic) over network interfaces such as ports or switch virtual interfaces (SVIs). This method benefits network traffic, because it is able to distribute the traffic more effectively without extending the data path. Load balancing improves the utilisation of network segments, which increases effective network range. Layer 3 Cisco switches support different layer protocols as well as advanced layer 3 IP and IPX (Internetwork Packet Exchange) protocol technology, which optimises network performance and scalability for networks which exhibit large and dynamic traffic patterns.

Layer 3 switches offer a high routing traffic performance between interfaces by identifying the availability of available interface on the network without relying on any single router. Likewise, they support 10/100/1000 Mb (Megabit) Ethernet, Gb (Gigabit) Ethernet, FEC (Fast Ethernet Channel), GEC (Gigabit Ethernet Channel) and BVI (Bridge Virtual interface). Layer 3 switches bolster the Virtual Local Network Interface (VLAN), which can combine any group of network

interfaces and segments within an inter-network into an autonomous user group, which appears as one LAN (Szigeti 2013:294-299).

3.6 Quality of Service (QoS) configuration technology

The proposed solution includes QoS configuration. A QoS technique permits the control of traffic over a network and guarantees the throughput of traffic applications in terms of time scale. QoS concerns the performance of the network traffic over several technologies, including 802.1 networks, IP-routed networks, Frame relay and Synchronous Optical Network (SONET) as seen from the user's perspective. Furthermore, QoS uses congestion management and avoidance techniques along with configuration and prioritises traffic based on its importance (Szigeti et al. 2013:1-9:83-85, Cisco 2014a:7-12 and Cisco 2016a:827).

The features of QoS are classified into the following functions: classifying and marking, policing, congestion management and avoidance. It offers a better and more reliable performance of network traffic service. QoS supports network management traffic and configuration methods, including a memory reservation, a dedicated bandwidth, a threshold, a throughput performance of network traffic (queues technology), shaping and sharing network traffic, and priority characteristics (Szigeti et al. 2013:31-32:83-109, Cisco 2014a:7-14 and Cisco 2016a:829). Employing QoS brands performance of network traffic more expectable and the utilisation of network bandwidth more effective.

Classification is the process of distinguishing one type of traffic from another by checking the fields in the packets or in the header. After traffic was classified, a traffic policy and marking were implemented to specify the bandwidth limits for each input and output traffic on the interfaces. Policing can decide on a packet-by-packet basis whether the packets are in or out of the scope of the profile and specify the action on the packets. The actions can be carried out by the marking function, including allowing the packets to pass without modification, dropping the packets and modifying the packets (marking down), such as assigning traffic to be matched with a DSCP value to allow the packets to pass through without being dropped. After traffic was classified and policing and marking had been implemented, a set of packets was processed in queues (input and output queues) at the specific bandwidth, and congestion was prevented by using congestion avoidance and Shared/Shaped Round Robin (SRR) features. Both the input and output queues use the SRR function to manage the guarantee of bandwidth. They use a congestion-avoidance mechanism called a weighted tail drop (WTD) to manage the lengths of queues and to deliver drop priority for specific classifications.

3.7 Parallel NIDPS technology

The solution made use of parallel NIDPS. Parallel NIDPS is a form of computation in which many NIDPS nodes work simultaneously, operating on the principle that the large incoming data can be divided into smaller sets, which are processed at the same time. Parallelism of NIDPS can occur at three general levels: the high-level processing node (entire system), the component level (specific tasks are isolated and parallelised) and the sub-component level parallelism (function within a specific task) (Wheeler and Fulp 2007). The handling of data can also be parallelised with traffic being split into separate streams to be processed by parallel nodes or components. This is data parallelism which can occur in various ways with the three general levels of parallelisation. In the novel architecture of this thesis, a parallel traffic was implemented through the use of queues (2 input queues and 4 output queues) on a switch virtual interface (SVI) where component level parallelism of NIDPS nodes was implemented with QoS configuration with the aim of improving NIDPS throughput performance and reducing NIDPS processor time (see Figure 3.5). The bandwidth capacity for each ingress queue was 50Mbps and each output queue was 25Mbps for each 100Mbps interface. For Gbps interfaces, the bandwidth was set to 500Mbps for each ingress queue and 250Mbps for each egress queue. Each ingress and egress buffer can be increased to its maximum interface bandwidth (100Mbps and/or 1Gbps). The NIDPS node was configured from a single-node NIDPS to a multi-node NIDPS. Each node was configured to check for a certain type of packet (e.g. UDP, TCP and ICMP) and was able to access discrete parts of a centralised, common rule base to order to carry out its task. The kernel buffer parameters for each NIDPS node was configured as each output queue rate.

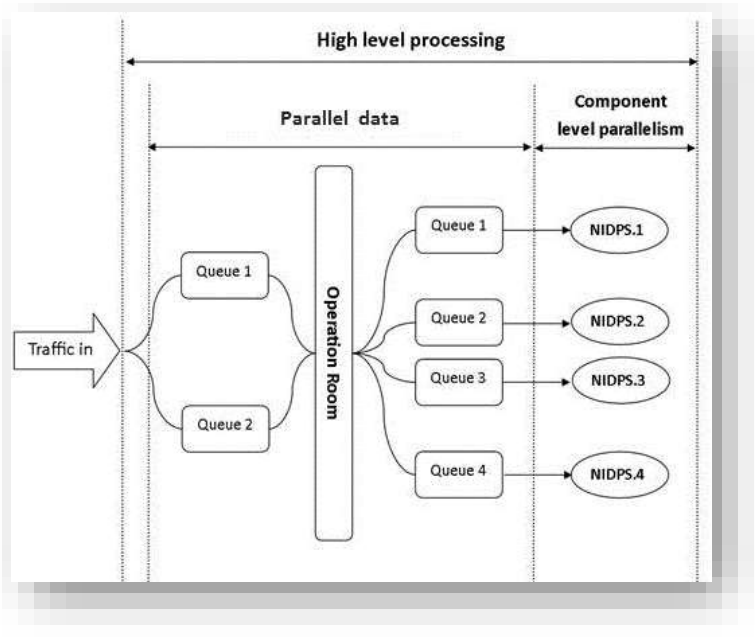


Figure 3. 5: High level parallel process.

The parallelisation of data (traffic) that was distributed through ingress and egress queues into critical and non-critical is viewed as multiple traffic parallelism (MTP) (Queues may be operating a different data (traffic)). This level parallelism of MT processing was linked to the component level parallelisation of the Snort NIDPS. Critical pre-processing of traffic is performed on queues to create particular groups of packets (threads) before the traffic is examined by a queue algorithm. Non-critical pre-processing occurred after the packets had been matched to queues. The NIDPS node component can be parallelised in either non-functional or a functional manner. Component level parallelism is defined as function parallelism of the NIDPS processing node. In component parallelism, individual components of NIDPS were isolated, and each output queue was given its own processing element.

3.8 NIDPS Methodology

As covered in chapter two, NIDPS methodology is divided into the following four categories: misuse/ signature-based; anomaly/ statistical-based; protocol analysis-based; and hybrid methodologies. In this research the technique of misuse detection was used to find known intrusions through signature detection.

In the experiments, a signature-based methodology is used to observe patterns inside the data packet. This method enables detection of various types of malicious traffic. Furthermore, this methodology can distinguish signatures in the headers of IP, UPD, TCP and ICMP. When a sought-after signature is found, alerts are activated and sent to system logs, databases, management teams or a trap. The NIDPS node was tested in a high-speed environment with large amounts of data. Three modes were configured to test the NIDPS performance: sniffing (analysis), passive (detection) and inline (prevention) method.

Analysis mode is employed to recognise and display the types of packets coming into the network. Various levels of detail can be displayed on the console, for instance, layer data attached to the packet in addition to TCP, UDP and ICMP header information. The detection system is capable of detecting suspicious activity and generating alerts based on recognised signatures and rules. Signature analysis is generally based on patterns inside the data packet. This technique aims to detect multiple kinds of attacks, such as the presence of scripts in packets destined for web services. Logging and alerts depend on the nature of what is detected inside the packets. If any suspicious activity is found inside a packet, the packet logs the malicious activity and/or generates an alert. Prevention mode intervenes to block intrusions that are detected before they reach the target. Prevention can be implemented through the use of signatures. Signatures contain IP addresses and can be considered safe or unsafe. The NIDPS contains known malicious packets in the form of a single or a set of

signatures. The detection mode is used to detect suspicious activity in the logs and generate alerts based on these signatures and rules. Furthermore, these technologies have proved their effectiveness in fighting flood type attacks (Burton, Baumrucker and Dubrawsky 2003, Wu, Schwab and Peckham 2008 and Weaver, Weaver and Farwood 2013).

To be effective, a NIDPS must see the entire network and must be placed at an appropriate point in the network. The NIDPS's sniffing mechanism is effectively configured and implemented at the network gateway, which delivers valuable information about traffic types and processing speed. In the experiments, the NIDPS was placed at a switch where QoS configuration and parallel technologies were implemented to enable a complete view and control of traffic to monitor, detect or prevent malicious packets in the network.

3.9 Experimental Design

3.9.1 The Experimental Stages

The overall experimental design can be seen as three stages:

A. Stage 1

Snort NIDPS was configured and tested in three modes: analysis (sniffer mode), detection (passive mode) and prevention (inline mode). Experiments were carried out to establish the amount of packet loss in increasing network traffic speed and volume for each mode. TCP/IP traffic was sent in 1 milliseconds (ms) and malicious packets in 1 microsecond (mSec) intervals. Both Windows and Linux operating systems were used.

B. Stage 2

The second set of experiments was conducted to establish the different packets dropped between (1) different operating systems, (2) different processor speeds and finally (3) different buffer sizes (speeds). Here, a virtual system was used.

C. Stage 3

An improved configuration based on QoS and parallel technology was developed. A third set of experiments was carried out to evaluate the solution.

3.9.2 The Experimental Testbed

A network was set up to serve as a model for the purpose of analysis and data acquisition (see Figure 3.6). It consisted of six physical stations and two virtual stations distributed as follows:

- Two physical check stations connected to the Cisco 3560/24P catalyst series switch; which supports QoS configuration;
- Four other physical stations connected to a Cisco switch 2950/16P; and
- Two VMware virtual machines running on one of the physical stations

Several tools, including both software and hardware, were used to carry out the research experiments, implementation and evaluation.

The software includes the following:

- Snort NIDPS software, installed on Windows operating systems 7, 8 and Linux OSs;
- Pcap tools (WinPcap and libpcap) to capture packets on OSs (Windows and Linux);
- NetScanPro tool to manage traffic in different time scales;
- Packet Generator tool to generate ICMP, UDP and TCP traffic at different speeds and values; and
- Flooder Packet, Tcpreply and Traceroute Packet tools to generate flood traffic and malicious UDP packets (threads) at high-load (65000KB) and high-speed (1ms and 1mSec).

The hardware includes the following:

- Cisco 3560/24P catalyst series switch, which supports QoS configuration. The system's capacity is shown as the following below :
  - 24 Fast Ethernet interface with 2 Small Form-Factor Pluggable (SFP)-based Gigabits Ethernet and 2 x2-based 10 Gigabit Ethernet ports Uplink;
  - 1 rack unit (RU) fixed-configuration, multilayer switch;
  - 32-Gbps forwarding bandwidth with maximum 128-Gbps wire rate, non-blocking switching fabric capacity;
  - 4 GB DDRAM with 64 MB Flash memory; and
  - Maximum buffer for each Fast Ethernet interface is 100Mb and for Gb interfaces is 1000Mb.

- Cisco Switch 2950, to implement Ethernet (port) channel (Ethchannel). The system's performance are:
  - 8 Fast Ethernet 10/100Mb with Uplinks 2x1G copper or 1G SFP;
  - 16-Gbps forwarding bandwidth with Maximum 32-Gbps switch bandwidth;
  - 2 GB with 32 MB Flash memory; and
  - Buffer for each interface is 100Mb.

- Computer network consisting of a minimum of six PCs with two VMware Virtual software machines (see Figure 3.6) with Intel Pentium® D CPU 2.2GHz, Intel® corei5 2.27GHz and Intel® corei7 2.40GHz.
- Network cables to connect the network.

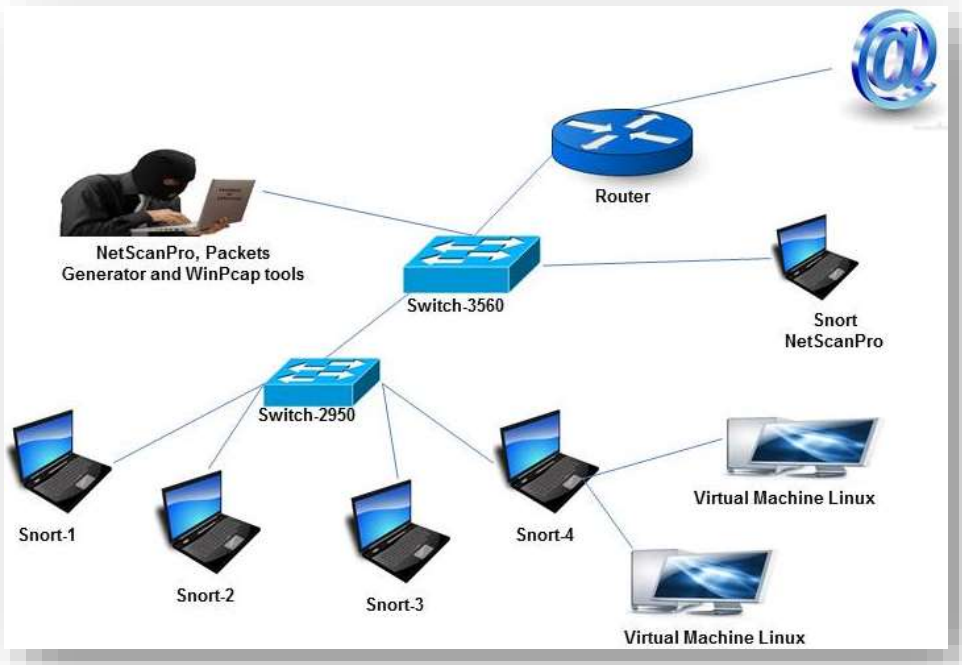


Figure 3. 6: Experiment network design.

3.9.3 Snort NIDPS tools and system requirements

Most of IDPS tools need configuration and additional software must be installed in order to run successfully. The following factors were considered in determining the tools to use with the Snort NIDPS:

- Accessible underlying signature database.

- Information about installing and setup configuration.
- If applicable, free of charge tools.
- The available tutorial and work related support.

The monitoring environment is an important component, as it helps analyse and protect or prevent networks from any malicious traffic or attacks. Monitoring environments are implemented by different OS interfaces, for example SSH, Apache, Linux and Windows. In this research, Terminal Service and Command Prompt interfaces were used to access the Snort NIDPS to meet experiment requirements.

Snort is one of most common monitoring NIDPS tools that safeguards a computer network and systems from any predictable attacks. It can be operated in either normal or special environments. Snort is not limited to a specific hardware, as it depends on the scalability of the network and systems. The technology of the host processor affects the application's speed in gathering and processing data packets. It also affects the network connection speed and data collection performance with regard to storage and logs.

For the research experiment design, the Switch Virtual Interface (SVI) and the host's NIC card needed to be the same speed as well as the network cable connections, otherwise, packets (traffic) may be dropped (missed). Furthermore, one of the switch's ports was used as a monitoring port, while the others were used for general serving, such as sending different traffic and speeds between hosts. An additional recommended requirement is to have a sufficient memory to grip a large amount of traffic (packets), which are exposed on the network detection engine.

Relevant libraries were required to install Snort NIDPS successfully in Windows and Linux OS. These included: WinPcap (Windows Packets capture) for Windows OSs; Pcap (libpcap-dev), PCRE (libpcre3-dev), and Libdnet (libdumbnet-dev) for Linux OSs; and DAQ (Data Acquisition) (Snort 2016). Furthermore, Snort NIDPS needed to be configured to run in different modes, such as sniffer, passive or inline mode. The following sections the associated tools used with Snort to create the experimental testbed are described.

3.9.4 Packets capture (Pcap) tool

The Pcap tool captures the traffic passing through the NIC card network; otherwise, traffic cannot be monitored. Pcap is the industry-standard tool for link-layer network access in OS environments, such as UNIX and Windows OSs. Pcap tools allow applications, such as open source

NIDPSs, to capture and resend network packets while avoiding the protocol stack. It has additional valuable features, such as packet filtering at the kernel level, statistics of network engine and also supports remote packet capture (Naveen, Natarajan and Srinivasan 2012 and Thanasekaran 2011 and WinPcap 2013).

WinPcap is a packet capture and filtering engine that is used with many commercial network tools, including open source, traffic and protocol analysers, network monitors, IDSs, IPSs, packets sniffers, traffic generators and network testers (Thanasekaran 2011and WinPcap 2013). Some such network tools, including e.g., NetScanPro, Nmap, Snort, Packets Flooder and ntop are identified and used throughout network community. WinPcap is a version of TcpDump and WinDump libraries, which are used to watch, diagnose and save network traffic according to various complex rules (WinPcap 2013).

Other tools required for Snort NIDPS in Unix OS include the following: PCRE (Perl-compatible regular expressions) and libdnet libraries. The PCRE library is a set of functions that execute systematic expression pattern matching. Libdnet simplifies transportable interfaces to numerous low-level networking routines, including manipulations of: network interfaces lookup; network IP address; kernel interfaces; IP header and Ethernet frame transmission (Mitra, Najjar and Bhuyan 2007 and Gullett 2012).

3.9.5 NetScanPro tool

NetScanPro is a tool that sends a certain type of traffic to specific networks and hosts. It offers a range of tools that gather internet information, monitor network and troubleshoot utilities for network professionals. These tools include the Packet Generator, Packet Flooder, Traceroute, Packet Capture, OS Fingerprinting, Ping-Enhanced, Ping Scanner and many more. Furthermore, it collects and captures IPv4, IPv6 and Hostnames, domain names, email addresses and URL information (NetScanToolsPro 2013). NetScanPro was designed to run on Win OS. The NetScanPro tool contains packet tools used in the research including the following:

1) Packet Flooder tool

The Packet Flooder tool is a network traffic generator. It can be sent different flooding packets (threads) to a target IPv4 or IPv6 address. It has control over the target interface and payload in the packets. It can send packets at a rate approaching nearly 100% of the interfaces bandwidth (wired internet). The flooder packets tool sends a malicious packet (thread) to a target as fast as a

computer's networking system will allow. The tool can send more than 65000 bytes per second (65000Bps) with up to 256 malicious packets (threads) in interval packet trips per microsecond (mSec).

2) Traceroute tool

The Traceroute tool is used to show the route of network packets that are traveling between a source computer and a target host. It can determine the upstream internet provider(s) that service a network connected device. Traceroute shows the individual routers that pass packets between computer and a target computer including the countries that are assigned to IP addresses along the route. The mechanism of tracing a network is based upon the ICMP protocol.

A very powerful additional mode is TCP Traceroute, which works by sending TCP packets to a valid network interface on the target, usually port 80, which means that this mode of traceroute will often penetrate firewalls from the outside. Although, when a traceroute attack starts sending malicious packets, a packet will be sent to a target address or a host with a TTL value of 1 (unless the user specifies otherwise); then if a timeout will occur from a responding system, another packet will be sent with a TTL of 2 and so forth. The TTL value is decremented on the header for each router in a network path. When the value of TTL is zero, the packet will discarded; then an ICMP message will be returned to the source with a signal that the time has been exceeded. When an ICMP packet's field is set to time exceeded, the IP address of the router will be placed in the IP header source field, which can be exploited by hackers. However, Traceroute is fully configurable, allowing users to control many parameters of the tracing process.

3) Packets Generation tool

The Packets Generation tool creates different types of traffic (packets), such as TCP, UDP, ICMP, CDP, ARP and RAW at different speeds (milliseconds) with having much control over most parameters of the IP and TCP/UDP headers. In the experiments, any packets that were sent used an actual source IP address in the IPv4 header. Packets were sent using the WinPcap driver. The tool can only send packets from WinPcap compatible interfaces, which are typically wired Ethernet interfaces, or through 802.11 wireless interfaces. The maximum size of packet can be send is 1 kilobyte per millisecond (1KBpms).

3.9.6 Tcpreply tool

Tcpreplay is a generator TCP traffic tool. It is employed in Linux OSs. It offers users an ability to test and capture traffic in network devices and systems. It can classify packets as clients or servers, rewrite the headers of Layers 2, 3 and 4 and replay the packets back onto the network and through other machines, such as switches, routers, and NIDPS, etc. Tcpreplay supports most of NIC modes (such as single and dual) for testing analysis, detection and prevention mechanisms. This tool can send traffic at speeds of more than 10Gbps.

3.9.7 Layer 2 and 3 Cisco Catalyst switches

Cisco Catalyst 3560 category belongs to layer 2 and 3 switches (Cisco 2016a:88). It provides support for IP-based functions, for example, rate limiting, access control lists (ACLs), QoS, IPv6 and advanced routing protocols. Policy and class enterprise features are supported by IP service software. Despite a packet's size and content, this switch provides the best effort services for each packet of network traffic. The packets are sent with no surety of delay bounds, reliability or throughput (Cisco 2016a:826). Using QoS configuration in this kind of switch can provide more control over traffic header and port parameters including setting queues, ACLs, VLANs, buffer, threshold, queue and traffic priority and bandwidth. Two other Cisco catalyst 2950 series switches were used to implement ether VLANs and channel techniques in order to investigate how to improve the throughput of data using parallel Snort NIDPS.

3.9.8 Experiment Performance Metrics

Performance metrics were used in the experiments to measure the capability of NIDPS to perform a certain task and to fit within the performance constraints. These metrics measure and evaluate the parameters that impact NIDPS performance. The following aspects were measured in the experiments.

1. Packet generation

The performance of TCP, UDP and ICMP protocols was measured when running over the IPv4 header. The WinPcap, Packets Generator tool and Flooder packets tools were used to vary the type of traffic and malicious packets (threads) in terms of IP header protocol (TCP, UDP and ICMP), speed, the number of packets and packet size.

2. Timing statistics

The Snort processor time includes total runtime of the packets processor as well as packet processing rates (Pkts).

3. Packets I/O totals and percentages

Various totals and percentages were used to measure the number of packets processed or not in the various experiments. The specific metrics used are shown in Table 3.1.

Table 3. 1: Snort performance metrics

NIDPS Mode	Performance metrics	Description	Sections
Analysis mode metrics	Packets received	Number of packets received by machines	Input / output total section
	Packets analysed	Percentage of packets analysed from total packets received	
	Packets dropped	Percentage of packets dropped from total packets received	
	Packets filtered	Packets filtered out and not handed to Snort for analysis	
	Packets outstanding	Number of the packets buffered waiting processing /or not processed	
	Packets injected	Injected packets are the result of active response, which can be configured for inline or passive modes.	
Detection mode metrics	Eth Packets received	Percentage of packets Eth received of total packets analysed	Breakdown by protocol section
	IP4 packets	Percentage of IP4 packets received of total Eth packets analysed	
	ICMP packets	Number of ICMP packets analysed of total Eth packets received	
	UDP packets	Number of UDP packets analysed of total Eth packets received	
	TCP packets	Number of TCP packets analysed of total Eth packets received	
	Alerts	Number of packet alerts of total packets analysed	Action status section
	logged	Number of packet logs of total packets analysed	
Prevention mode metrics	Block	Number of packets blocked, dropped or rejected of total packets analysed	Verdicts section

4. Protocol statistics

All traffic for all protocols decoded by Snort are summarised in the Snort breakdown section (see Table 3.1), which includes categories, such as Eth (Ethernet interfaces), VLAN, IP4, Frag (Fragmented packages), ICMP, UDP, TCP and others.

3.9.9 Experiments Conducted

The experiments carried out are listed in Table 3.2, along with their purpose.

Table 3. 2: Experiments conducted

Number of experiments	Purpose	Section in Thesis
Stage 1 - Experiments to analyse the problem		
Experiment 1 to 4	Test NIDPS analysis performance at high-speed and heavy traffic	4.3
Experiment 5 to 8	Test NIDPS detection performance at high-speed traffic	4.4
Experiment 9 -11	Test NIDPS prevention performance at high-speed traffic	4.5
Stage 2 - Experiments to investigate reason of the problem and support stage 1.		
Experiment 12	To test NIDPS performance under different Oss, different Buffer size and different processor (speed).	4.6
Stage 3 - Experiments to evaluate the solution		
Experiment 13	Test the evaluation solution for NIDPS analysis performance	6.2.2
Experiment 14-18	Test the evaluation solution for NIDPS detection performance	6.2.3 and 6.2.4
Experiment 19-20	Test the evaluation solution for NIDPS prevention performance	6.2.5
Experiment 21	Show how parallel technology can benefit NIDPS through QoS	6.3.1
Experiment 22	Show NIDPS architecture performance under more than 8 Gbps traffic speed.	6.3.2

3.10 Conclusion

This chapter provided information and details about the methodology and experimental design of this research and described the software and hardware that was used to conduct the experiments.

CHAPTER 4: RESEARCH PROBLEM ANALYSIS

4.1 Introduction

In this chapter, analysis of the research problem is provided. The analysis was carried out through experimentation which was termed stage 1 experimentation in this research. The results are presented of the stage 1 and stage 2 experiments. The list of experiments is given in section 4.2. The stage 1 experiments were carried out to establish the level of packet loss using three NIDPS modes: analysis; detection; and prevention. The results for analysis mode are presented in section 4.3, for detection mode in section 4.4 and prevention mode in section 4.5. Stage 2 experiments tested NIDPS performance under different OSs, buffer size and processor speeds and the results are presented in section 4.6. Overall the research found that the NIDPS performance decreases in a high-speed and high-volume traffic in all three modes and tasks.

4.2 Summary of Experiments carried out

The experiments carried out are shown in Table 4.1, along with their purpose.

Table 4. 1: Summary of experiments.

Experiment Number	Purpose	Section in Thesis
Analysis Mode Experiments		
1	to show Snort-NIDPS analysis performance under heavy traffic	4.3.1
2	to show Snort-NIDPS analysis performance under high-speed traffic	4.3.2
3	to show Snort-NIDPS analysis performance under large data traffic	4.3.3
4	to show Snort-NIDPS analysis performance under heavy traffic and high-speed	4.3.4
Detection Mode Experiments		
5	to show Snort-NIDPS performance detection under high-speed ICMP traffic	4.4.1
6	to show Snort-NIDPS performance detection under high-speed UDP traffic	4.4.2
7	to show Snort-NIDPS performance detection under high-speed TCP traffic	4.4.3
8	to show Snort-NIDPS performance detection under a heavy and high-speed malicious traffic	4.4.4
Prevention Mode Experiments		
9	to show Snort-NIDPS performance prevention under high-speed IP (ICMP and UDP) traffic	4.5.1
10	to show Snort-NIDPS performance prevention under high-speed TCP traffic	4.5.2
11	to show Snort-NIDPS performance preventing under a heavy and high-speed malicious traffic	4.5.3

Different tasks Experiments		
12	To show Snort-NIDPS performance under different OSs, buffer size and processor speeds.	4.6

4.3 NIDPS's Performance analysis-mode (sniffer mode)

Here, Snort NIDPS has been configured to analysis or Sniffer mode. The following metrics were recorded: the number of packets received of the total packets sent; the number of packets analysed of the total packets received; the number of packets dropped of the total packets received; the number of packets rejected of the total packets received; and the number of packets outstanding of the total packets received. Specific results are given in the following sections.

4.3.1 Experiments 1s: Testing Snort NIDPS under heavy traffic

The transmission rate of packets was kept to the same speed (1ms intervals) to obtain a fair analysis of different numbers of packets. For this experiment, three (3) consecutive tests were run to test TCP, UDP and ICMP headers. For each test, the number of the packets sent was increased. The packets sent were 1024 bytes (each packet carried 1KB). NetScanPro and WinPcap tools were used to manage traffic through the network and the Packet Generator tool was used to send different numbers of packets and types of traffic (TCP, UDP or ICMP) at the same speed (1ms intervals) through the network and hosts.

4.3.1.1 Experiment 1.1: Snort reactions to TCP header under heavy traffic.

Table 4. 2: Snort reaction to TCP header.

No of Test	Packets sent	Packets received	Packets analysed	Packets filtered	Packets injected	Packets dropped	Packets outstanding	Percentage analysed packets	Percentage dropped packets	Percentage outstanding packers
1	100	105	105	0	0	0	0	100.00%	0.00%	0.00%
2	200	202	202	0	0	0	0	100.00%	0.00%	0.00%
3	400	402	402	0	0	0	0	100.00%	0.00%	0.00%
4	800	805	538	0	0	266	267	66.832%	24.837%	33.168%
5	1600	1,606	970	0	0	636	636	60.399%	28.368%	39.601%
6	3200	3,208	964	0	0	2,241	2241	30.050%	41.127%	69.950%
7	6400	6,417	1,418	0	0	4,998	4999	22.098%	43.784%	77.902%
8	100000	100067	13169	0	0	86898	86898	13.160%	46.478%	86.840%
9	200000	200144	25383	0	0	174759	174761	12.682%	46.614%	87.318%

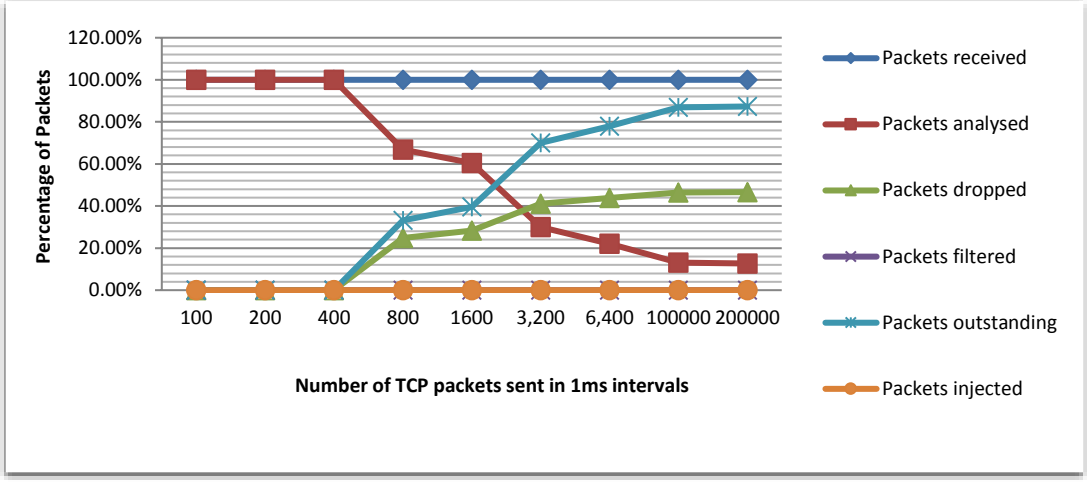


Figure 4. 1: Snort reaction to TCP header under heavy traffic.

4.3.1.2 Experiment 1.2: Snort reactions to UDP header under heavy traffic.

Table 4. 3: Snort reactions to UDP header.

No of test	Packets sent	Packets received	Packets analysed	Packets filtered	Packets injected	Packets dropped	Packets outstanding	Percentages analysed packets	Percentages packets dropped	Percentages packet outstanding
1	100	105	105	0	0	0	0	100.00%	0%	0%
2	200	210	161	0	0	49	49	76.667%	18.919%	23.333%
3	400	406	151	0	0	255	255	37.192%	38.578%	62.808%
4	800	813	273	0	0	539	540	33.579%	39.867%	66.421%
5	1600	1607	266	0	0	1341	1341	16.553%	45.488%	83.447%
6	3200	3219	390	0	0	2829	2829	12.116%	46.776%	87.884%
7	6400	6420	603	0	0	5817	5817	9.393%	47.536%	90.607%
8	100000	100174	7246	0	0	92928	92928	7.233%	48.124%	92.767%
9	200000	200357	14466	0	0	185885	185891	7.220%	48.127%	92.780%

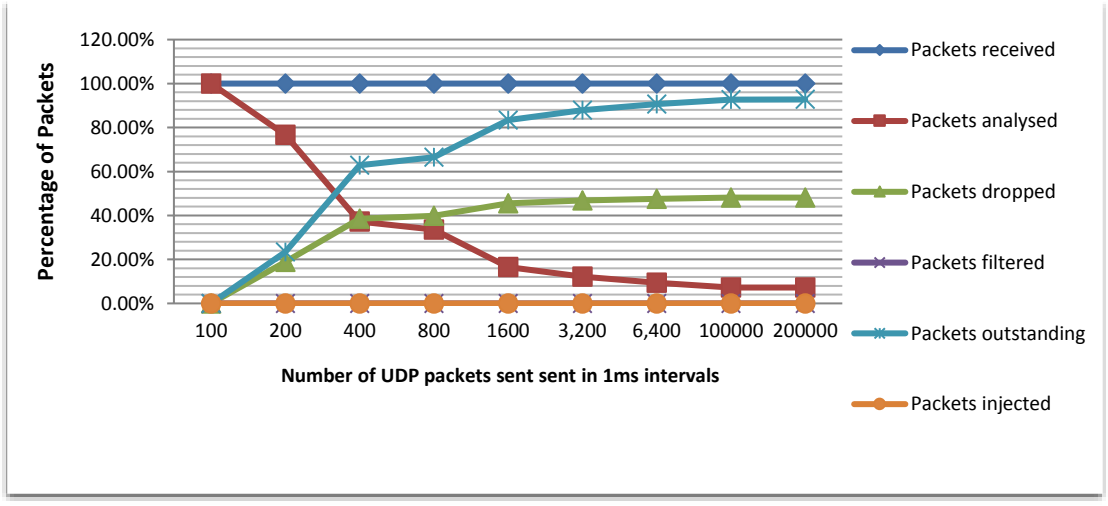


Figure 4. 2: Snort reaction to UDP header under heavy traffic.

4.3.1.3 Experiment 1.3: Snort reactions to ICMP header under heavy traffic.

Table 4. 4: Snort reactions to ICMP header.

No of test	Packets sent	Packets received	Packets analysed	Packets filtered	Packets injected	Packets dropped	Packets outstanding	Percentages packets analysed	Percentages packets outstanding	Percentages packet dropped
1	100	105	105	0	0	0	0	100.00%	0.00%	0.00%
2	200	206	206	0	0	0	0	100.00%	0.00%	0.00%
3	400	403	296	0	0	107	107	73.449%	26.551%	20.980%
4	800	804	370	0	0	434	434	46.020%	53.980%	35.057%
5	1600	1605	527	0	0	1,078	1078	32.835%	67.165%	40.179%
6	3200	3,212	752	0	0	2,458	2460	23.412%	76.588%	43.351%
7	6400	6,417	993	0	0	5,424	5424	15.475%	84.525%	45.807%
8	12800	12812	1905	0	0	10907	10907	14.869%	85.131%	45.984%
9	100000	100061	12043	0	0	88018	88018	12.036%	87.964%	46.798%
10	200000	200140	23805	0	0	176335	176335	11.894%	88.106%	46.838%

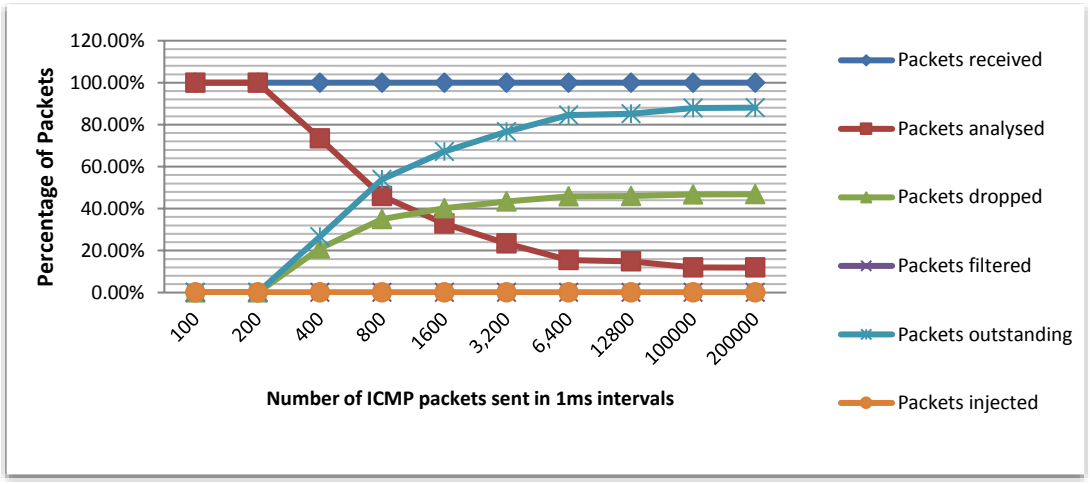


Figure 4. 3: Snort reaction to ICMP header under heavy traffic.

As demonstrated by the results shown in Figures 4.1, 4.2, and 4.3, all the packets that were sent reached the wire. Figures show that when 100 and 200 packets were sent at speed 1ms, Snort analysed 100% of the total packets that it received. As the number of packets was increased, Snort started dropping packets or leaving packets outstanding (see Figures 4.1, 4.2 and 4.3). Figures also show that as the number of packets increases, more packets are dropped and left outstanding .The experiments show that dropped packets start to occur from 400 to 200000 at speed 1ms, Snort's efficiency dropped more than 46 percent for (TCP) and (ICMP) headers (see Tables 4.2 and 4.4), and more than 43 percent for (UDP) headers (see Table 4.3). More than 87 percent of packets were outstanding for (TCP) headers (see Table 4.2), more than 77 percent for (UDP) headers (see Table 4.3) and more than 88 percent for ICMP headers (see Table 4.4), and less than 13, 23 and 12 percent of the total packets received for (TCP, UDP and ICMP respectively) were analysed (see Tables 4.2,

4.3 and 4.4,). The experiments show that Snort performance analysis has been affected when the value of the traffic is increased.

4.3.2 Experiments 2s: Testing Snort NIDPS under high-speed traffic

The number of the packets was kept to the same value, 200,000, for a fair analysis between different speeds. Here, three (3) consecutive tests were run for (TCP, UDP and ICMP) headers, for each test, the speed at which the packets were sent was increased.

4.3.2.1 Experiment 2.1: Snort reactions to ICMP header under high-speed traffic.

Table 4. 5: Snort reaction to ICMP header.

Packets trip time	Packets received	Packets analysed	Packet filtered	Packets injected	Packets dropped	Packets outstanding	Percentages packets analysed	Percentages packets outstanding	Percentages packets dropped
32ms interval	203622	203622	0	0	0	0	100.00%	0.00%	0.00%
16ms interval	201757	201757	0	0	0	0	100.00%	0.00%	0.00%
8ms interval	201266	188205	0	0	13060	13061	93.511%	6.489%	6.094%
4ms interval	200420	94154	0	0	106266	106266	46.978%	53.022%	34.650%
2ms interval	200221	47130	0	0	153090	153091	23.539%	76.461%	43.330%
1ms interval	200131	23793	0	0	176338	176338	11.889%	88.111%	46.840%
0.5ms interval	199890	2364	0	0	197526	197526	1.183%	98.817%	49.703%

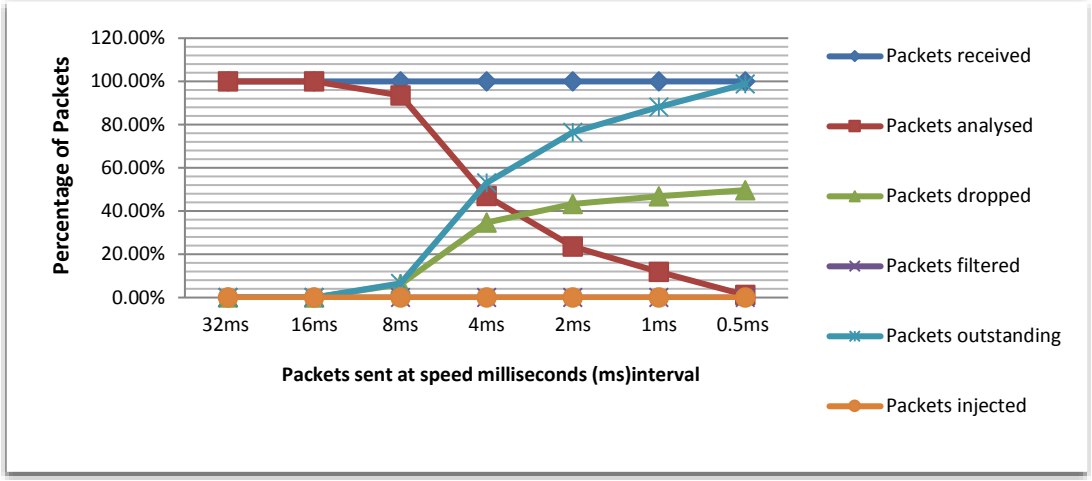


Figure 4. 4: Snort reaction to ICMP header under high-speed traffic.

4.3.2.2 Experiment 2.2: Snort reactions to UDP header under high-speed traffic.

Table 4. 6: Snort reaction to UDP header.

Packets trip time	Packets received	Packets analysed	Packets filtered	Packets injected	Packets dropped	Packets outstanding	Percentages packet analysed	Percentages packet outstanding	Percentages packet dropped
32ms interval	201099	200601	0	0	0	0	100.00%	0.00%	0.00%
16ms Interval	201067	201067	0	0	0	0	100.00%	0.00%	0.00%
8ms interval	200997	200997	0	0	0	0	100.00%	0.00%	0.00%
4ms interval	200922	108799	0	0	92123	92123	54.149%	45.851%	31.436%
3ms interval	200715	81950	0	0	118765	118765	40.829%	59.171%	37.174%
2ms interval	200426	54730	0	0	145694	145697	27.307%	72.693%	42.09%
1ms interval	200225	27831	0	0	172394	172394	13.600%	86.400%	46.26%
0.5ms interval	200031	3385	0	0	196644	196648	1.692%	98.308%	49.58%

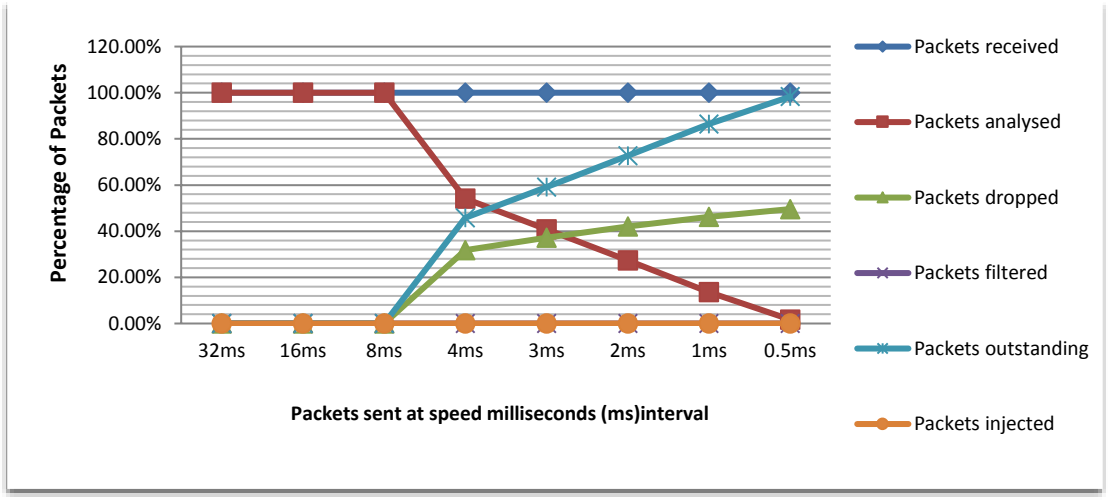


Figure 4. 5: Snort reaction to UDP header under high-speed traffic.

4.3.2.3 Experiment 2.3: Snort reactions to TCP header under high-speed traffic.

Table 4. 7: Snort reaction to TCP header.

Packets Time trip	Packets received	Packets analysed	Packets filtered	Packets injected	Packets dropped	Packets outstanding	Percentages packet analysed	Percentages packet outstanding	Percentages packet dropped
32ms interval	200320	200420	0	0	0	0	100.00%	0%	0%
16ms interval	200350	200741	0	0	0	0	100.00%	0%	0%
8ms interval	200004	200000	0	0	0	4	99.998%	0.002%	0%
4ms interval	200469	104070	0	0	96397	96399	51.913%	48.087%	32.472%
3ms interval	200382	78141	0	0	122238	122241	38.996%	61.004%	37.889%
2ms Interval	200327	52458	0	0	147869	147869	26.186%	73.814%	42.467%
1ms interval	200147	26734	0	0	173413	173413	13.357%	86.643%	46.422%
0.5ms interval	200104	13759	0	0	186345	186345	6.876%	93.124%	48.220%

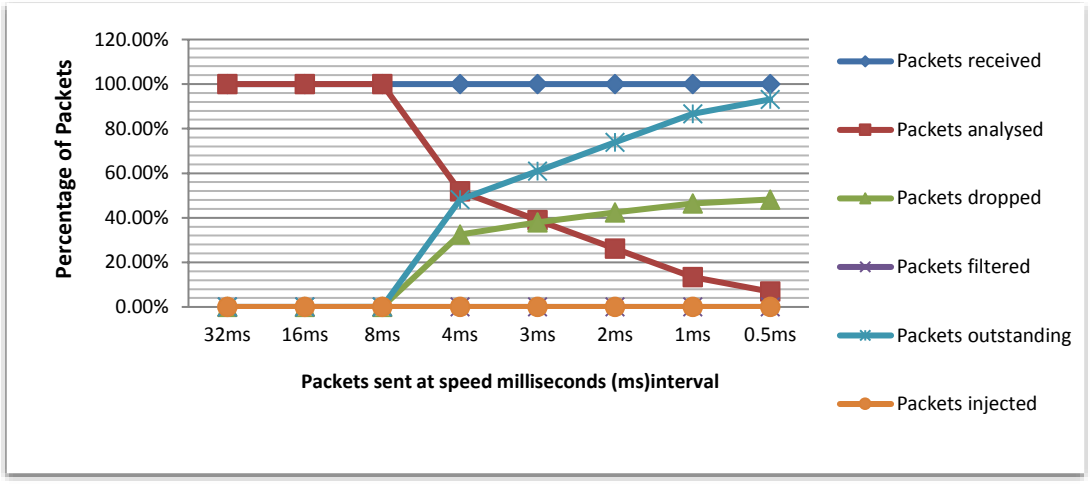


Figure 4. 6: Snort reaction to TCP header under high-speed traffic.

The results shown in Figures 4.4, 4.5 and 4.6 reveal that Snort initially was analysed all packets that reached the wire. As the speed increased, Snort started dropping the packets and some were left outstanding. Figures 4.4, 4.5 and 4.6 show that the number of packets dropped and the number of packets outstanding increase as the speeds increase. The experiments show that Snort dropped more than 49, 49 and 48 percent, and left outstanding more than 98, 98 and 93 percent. So Snort analysed less than 2, 2 and 7 percent of the total packets analysed for ICMP, UDP and TCP respectively as the transmission interval decreased from 4ms to 0.5ms (see Tables 4.5, 4.6 and 4.7). The experiments show that Snort's analysis performance reduced while speed of transmission increased.

4.3.3 Experiments 3s: Test Snort NIDPS under large packets

For this experiment the number of the packets was kept to the same value, 5000, and the same speed (rate of transmission 2ms per packet) for fair analysis between different sizes and lengths of packets. Here, three (3) consecutive tests were run; for each test, the size of each packet sent "Len" was increased, starting from 1 byte, 400bytes, 800bytes to 1Kbyte.

4.3.3.1 Experiment 3.1: Snort reactions to ICMP header under large packets.

Table 4. 8: Snort reaction to ICMP header.

Packets size "weight"	Packets received	Packets analysed	Packets dropped	Packets filtered	Packets injected	Packets outstanding	Percentages packet analysed	Percentages packet outstanding	Percentages packet dropped
0 bytes	10011	10011	0	0	0	0	100.00%	0.00%	0.00%
400 bytes	5018	1400	3617	0	0	3618	27.930%	72.070%	41.887%

800 bytes	5023	636	4387	0	0	4387	12.661%	87.339%	46.615%
1024 bytes (1kB)	5012	507	4505	0	0	4505	10.036%	89.964%	47.333%

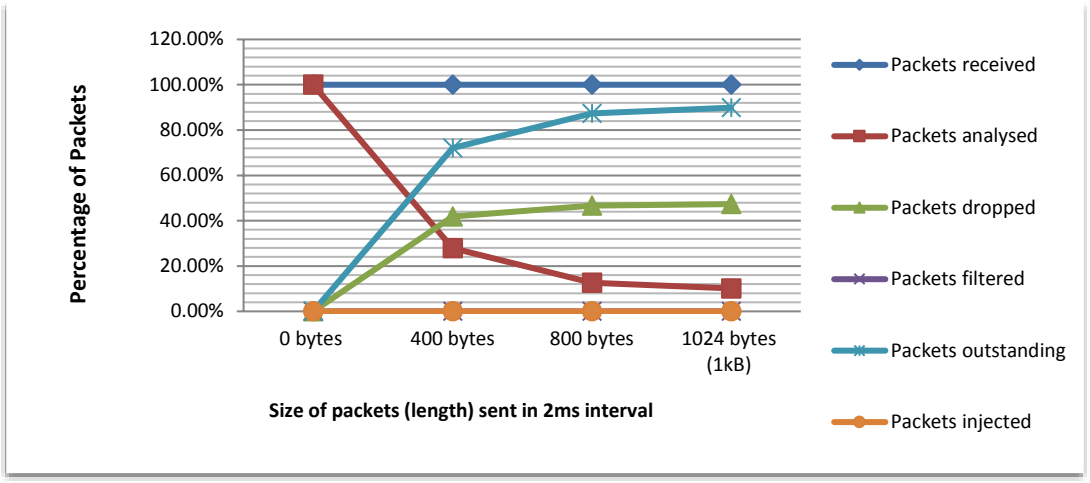


Figure 4. 7: Snort reaction to ICMP header under large packets.

4.3.3.2 Experiment 3.2: Snort reactions to UDP header under large packets.

Table 4. 9: Snort reaction to UDP header.

packets size "weight"	Packets received	Packets analysed	Packets dropped	Packets filtered	Packets injected	Packets outstanding	Percentages packet analysed	Percentages packet outstanding	Percentages packet dropped
0 bytes	5023	5022	0	0	0	1	99.980%	0.020%	0%
400 bytes	5019	2649	2366	0	0	2370	52.779%	47.221%	32.038%
800 bytes	5019	1393	3625	0	0	3626	27.755%	72.245%	41.937%
1024(1KB) bytes	5019	963	4055	0	1	4056	19.187%	80.813%	44.688%

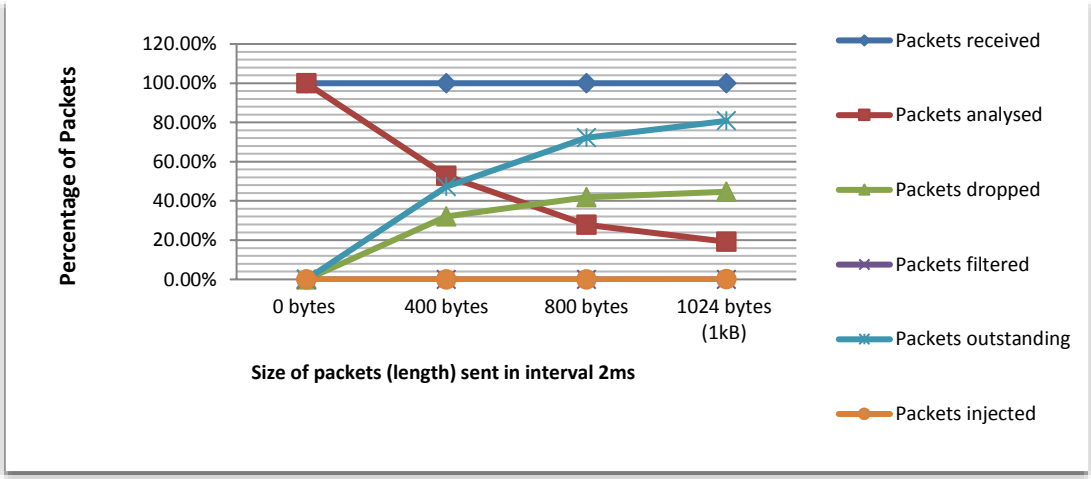


Figure 4. 8: Snort reaction to UDP header under large packets.

4.3.3.3 Experiment 3.3: Snort reactions to TCP header under large packets.

Table 4. 10: Snort reaction to TCP header.

Packets size "weight"	Packets received	Packets analysed	Packets dropped	Packets filtered	Packets injected	Packets outstanding	Percentages packet analysed	Percentages packet outstanding	Percentages packet dropped
0 bytes	5019	5019	0	0	0	0	100.00%	0.00%	0.00%
400 bytes	5012	2070	2942	0	0	2942	41.310%	58.690%	36.984%
800 bytes	5013	1359	3654	0	1	3654	27.115%	72.885%	42.148%
1024(1kb) bytes	5006	920	4086	0	0	4086	18.379%	81.621%	44.940%

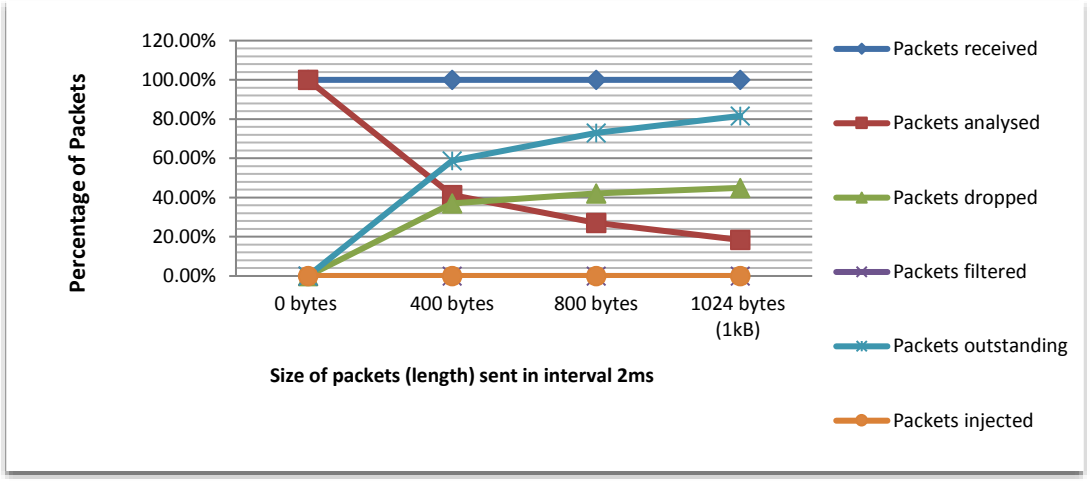


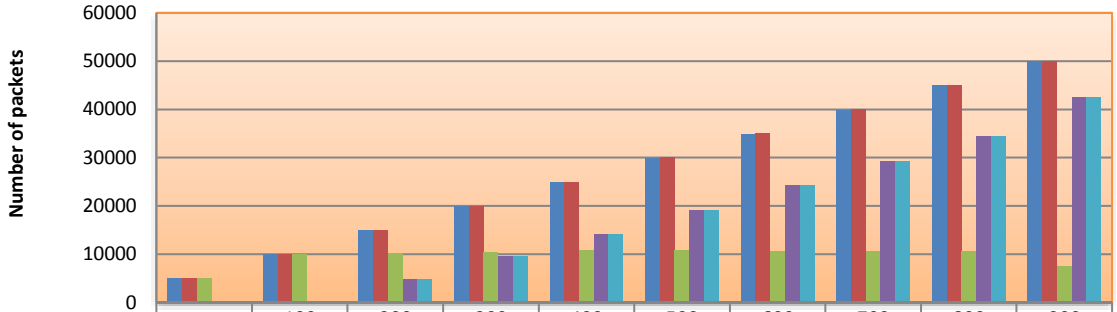
Figure 4. 9: Snort reaction to TCP header under large packets.

As shown in Figures 4.7, 4.8 and 4.9, Snort initially analysed every single packet that reached the wire. As the size of packets was increased, Snort started dropping and leaving the packets outstanding. Also the figures show that while the size of packets (Len) was increased, the number of packets outstanding and packets dropped increased as well. The experiments showed that Snort dropped more than 47 percent of ICMP packets, and more than 44 percent of UDP and TCP packets of the total packets received as the size of packet changed from 400B to 1KB. Left outstanding was more than 89 percent of ICMP, 80 percent of UDP and 81 percent of TCP packets of the total packets received (see Tables 4.8, 4.9 and 4.10). The experiments show that Snort analysis performance was affected while increasing the size (Len) of packets.

4.3.4 Experiment 4: Testing Snort NIDPS under heavy traffic and high-speed

In this experiment, IP traffic has been sent with different values, speed and size. For each test the number of packets, speed traffic and size of each packet was increased.

As shown in Figure 4.10, the experiment demonstrated that, as the volume and speed of traffic increased, the number of packets dropped and outstanding increased drastically as well. Snort's analysis rate decreases as traffic and speed increase in a computer network.



	1 B/10ms	100 Bs/9Ms	200 Bs/8ms	300 Bs/7ms	400 Bs/6ms	500 Bs/5ms	600 Bs/4ms	700 Bs/3ms	800 Bs/2ms	900 Bs/1ms
Packets sent	5000	10000	15000	20000	25000	30000	35000	40000	45000	50000
Packets received	5014	10020	15031	20042	25049	30064	35066	40081	45087	50130
Packets analysed	5014	10018	10179	10424	10789	10820	10736	10711	10681	7637
Packets dropped	0	0	4852	9618	14260	19244	24330	29370	34406	42493
Packets outstanding	0	2	4852	9612	14262	19245	24330	29372	34408	42496

Figure 4. 10: Snort reactions under heavy traffic and high-speed.

4.4 NIDPS's Performance detection-mode (passive-mode)

Here, Snort NIDPS has been configured to NIDPS detection mode (NID-mode). This mode is used to detect malicious traffic. In this part of the experiments, the second and third sections of the Snort metrics has been used, such as Snort's breakdown by protocol (i.e., the number of packets analysed of the total packets received, the number of Ethernet (Eth) packets received of the total packets analysed, the number of IP packets received of the total Eth packets received, and the number of TCP, UDP and ICMP packets analysed) and Snort's action statistics (i.e., the number of packet alerts of the total TCP/IP packets analysed and the total packets logged of the total TCP/IP packets analysed). The experiments were conducted to test Snort NID-mode performance reaction to detect (TCP/IP) headers and malicious packets (threads) under high-speed traffic.

4.4.1 Experiment 5: Snort NIDPS reactions to alerts and logs with ICMP header

In this experiment, more than 1 million IP/ICMP packets have been sent at different speeds (10ms, 5ms and 1ms interval). The size of each packet was carried out 1KB. The rule below was used which requires that Snort will alert and detect any ICMP packets from any sources and ports to any destinations and ports.

Alert icmp any any ->any any (msg:"Detect ICMP Packets"; sid: 100001 ;).

Table 4. 11: Snort reaction to ICMP header.

Traffic speed per millisecond	Machine Packets received	% packet analysed	Eth packets received of the total packets analysed	ICMP packets analysed	TCP packet analysed	UDP packets analysed	Packets alert	Packets logged	% packet alerts	% packet logs
10ms	100%	4.300%	100%	1874	0	44459	1874	1874	100%	100%
5ms	100%	1.120%	100%	345	0	13463	231	231	66.96 %	66.96 %
1ms	100%	0.141%	100%	730	0	1144	405	405	55.47 %	55.47 %

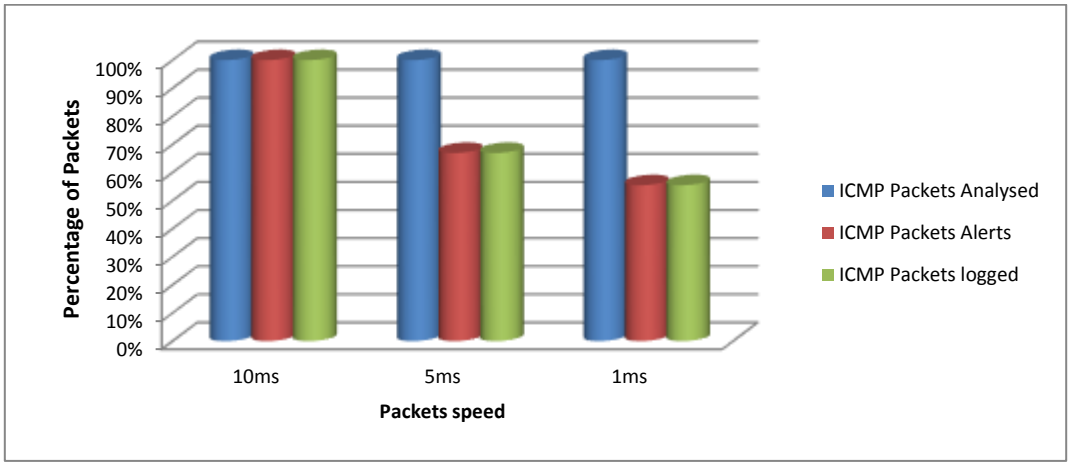


Figure 4. 11: ICMP packets detection.

As the results show in Figure 4.11, Snort analysed every packet that reached the wire. When ICMP traffic was sent at 10ms, Snort alerted and logged nearly 100% of the total ICMP packets analysed (see Table 4.10). As the speed increased from 10ms to 1ms, Snort started missing alerts and logged packets. Also, Figure 4.11 shows that the number of missed alerts increased when the speed increased. The experiment shows that Snort detected 55.47 % of the total ICMP packets that it analysed (see Table 4.11).

4.4.2 Experiment 6: Snort NIDPS reactions to alerts and logs with UDP header

In this experiment, more than 1 million IP/UDP packets were sent at different speeds (10ms, 5ms, 3ms and 1ms), the packet size was 1KB and the following rule was written to allow Snort to

detect any UDP packets from any sources to any destination address and to any source and destination ports:

Alert udp any any ->any any (msg:"Detect UDP Packets"; sid: 100002 ;).

Table 4. 12: Snort reaction to UDP header.

Traffic Speed Per millisecond	Machine Packets Received	% Packets Analysed	Eth packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	Packet alerts	packets logged	% Packets alerts	% Packets logged
10ms	100%	11.293%	100%	0	0	7854	7798	7798	99.28%	99.28%
5ms	100%	3.128%	100%	0	0	2958	2896	2896	97.90%	97.90%
3ms	100%	1.274%	100%	0	0	1358	946	946	69.66%	69.66%
1ms	100%	1.006%	100%	0	0	65	30	30	46.15%	46.14%

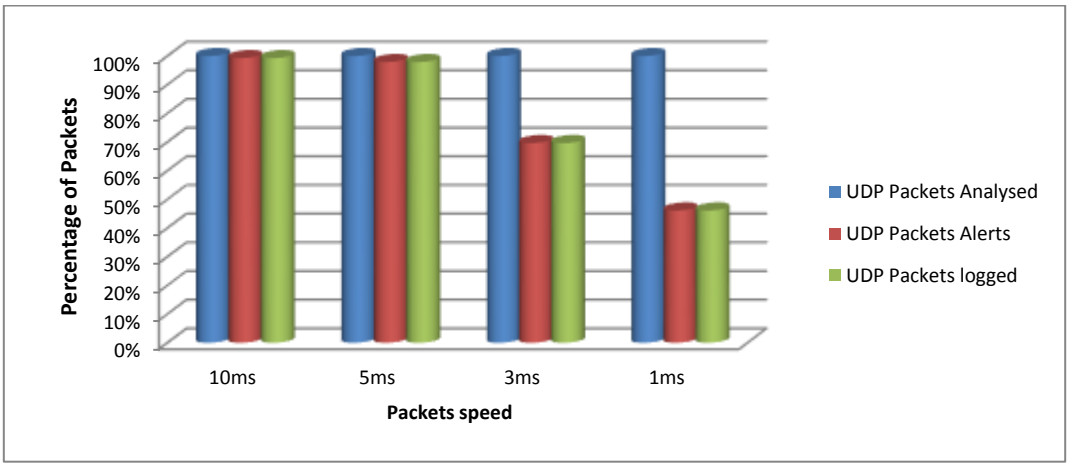


Figure 4. 12: UDP packets detection.

As shown in Figure 4.12, when UDP traffic was sent at a speed of 10ms, Snort alerted and logged nearly 100% of the total UDP packets that it analysed (see Table 4.11). When the traffic's speed increased to 5ms, Snort detected 97.90% of the total UDP packets analysed (see Table 4.12). Figure 4.12 shows that, as the speed increased, missed alerts and logs also increased. This experiment shows that Snort detected 46.14% of the total UDP packets that it analysed (see Table 4.12).

4.4.3 Experiment 7: Snort NIDPS reactions to alerts and logs with TCP header

Here, more than 1 million TCP/IP packets were sent at different speeds (10ms, 5ms and 1ms). Each packet was 1KB in size. The following rule was made to allow Snort to detect any TCP packets from any sources to any destinations, from and to any ports:

Alert tcp any any ->any any (msg:"Detect tcp Packets"; sid: 100003)

Table 4. 13: Snort reaction to TCP header.

Traffic Speed per millisecond	Machine packets received	% packets analysed	Eth packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	Packets alerts	Packets logged	% Packets alerts	% Packets logged
10ms	100%	51.567%	100%	128	51070	25	5170	5170	100%	100%
5ms	100%	3.122%	100%	110	33113	31	33113	33113	100%	100%
1ms	100%	0.981%	100%	0	14622	249	14622	14622	100%	100%

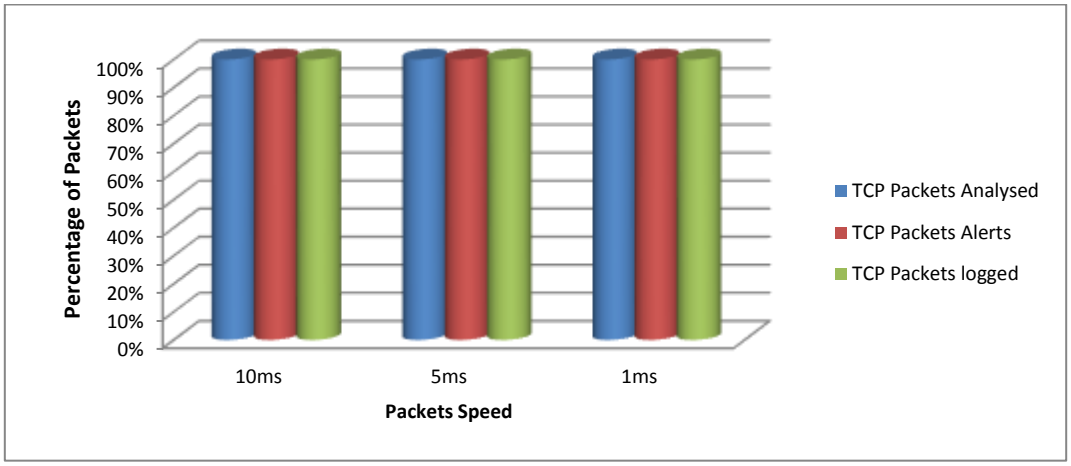


Figure 4. 13: TCP packets detection.

As shown in Figure 4.13, Snort analysed all packets that reached the system. The experiment shows that Snort detected all TCP packets that it analysed, even if the speed increased (see Table 4.13). This effectiveness occurred because TCP does not send the next packet until it receives an acknowledgement that the previous package has been received. For example, when a device sends a TCP packet at a specific time, it waits for an acknowledgement for a certain period, and the transmission will be paused until the acknowledgement is received. These acknowledgements make the TCP packet slower than the UDP and ICMP packets.

4.4.4 Experiment 8: Snort NIDPS reactions to detecting malicious packet (Threads) in high-speed traffic

In this experiment, WinPcap and Flooder packet tools were used to send flood traffic with malicious UDP packets (255 threads per 1mSec) to specific hosts or networks at different speeds (see Table 4.14). The following rule allowed Snort to alert and log any UDP threads or malicious packets that contain the variables 'abcdef' and time to live (TTL) 128 that comes from any source and port address and goes to any destination address and ports:

Alert udp any any ->any any (msg: "Detect Malicious UDP Packets"; ttl: 128; content:|' 61 62 63 64 65 66 '|; Sid: 100004 ;)

This experiment is different from the previous ones. The previous experiments tried to detect headers, such as TCP, UDP and ICMP. The system received the TCP, UDP and ICMP packets at different speeds, but in this experiment, flood traffic was sent in different bandwidths (speeds) with 255 malicious UDP packets (threads) in interval packets with a delay of 1 microsecond (1 mSec). Snort was set up to detect only the malicious UDP threads by using two conditions of additional rules (TTL and content). These two key rules will detect any UDP malicious packet that is matched in order to determine that the TTL value is equal to 128 and to determine if a data pattern inside the malicious packet has variables ('abcdef'). The hexadecimal number ('61 62 63 64 65 66'), which the rule contained, is equal to the ASCII characters ('a b c d e f').

Table 4. 14: Snort reaction to udp malicious packets.

flood traffic (Byte PerSeconds) With 255 UDP malicious packets in (1mSec)	Total Eth received Of the total Packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	Malicious packets Alerts	Malicious packets logged	% Packets alerts	% Packets logged
16 Bps	100%	0	0	9868	9820	9820	99.51%	99.51%
32 Bps	100%	0	0	8702	8654	8654	99.44%	99.44%
200 Bps	100%	0	0	7166	7083	7083	98.84%	98.84%
1200 Bps	100%	0	0	6024	5854	5854	97.17%	97.17%
4800 Bps	100%	0	0	2876	1421	1421	49.40%	49.40%
60000 Bps	100%	0	0	7560	2810	2810	35.75%	35.75%

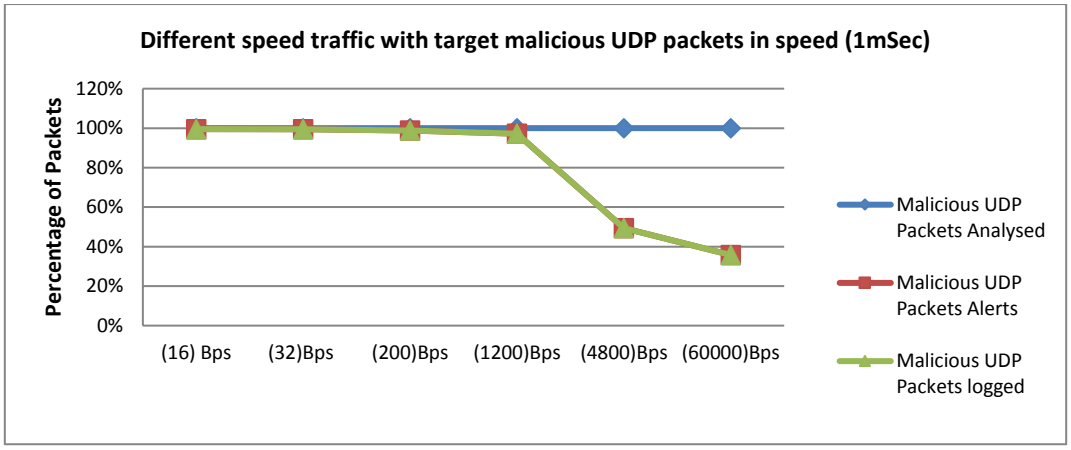


Figure 4. 14: Malicious packets detection.

As shown in Figure 4.14, Snort initially analysed every packet that reached the wire. When malicious UDP packets were sent at a speed of 1 mSec and flood traffic at 16 bytes per second (Bps), Snort alerted and logged more than 99% of the total UDP packets that it analysed. As the flood traffic (speed) was increased to 200, 1200, 4800 and 60000 bytes per second (Bps), Snort alerted and logged packets to a decreasing degree, respectively, at 98.84, 97.17, 49.40 and 35.75% of the total malicious packets analysed (see Table 4.14). Figure 4.14 shows that the number of missed malicious packet alerts increased when the speed increased. The experiment shows that, when the speed was 60000 Bps, Snort only detected nearly 35% of the malicious packets analysed (see Table 4.14).

4.5 NIDPS's performance prevention mode (inline-mode)

Snort was configured to inline mode (NIP-mode) on Linux OS, because Win OS does not support inline mode. In this experiment, the fourth (4th) processor section action has been used on Snort, and the metrics considered are: the number of packets dropped of the total packets analysed; the number of packets blocked of the total packets analysed; and the number of packets rejected of the total packets analysed.

4.5.1 Experiment 9: Snort NIDPs reaction to drop (prevent) IP (ICMP/UDP) header

In this experiment, IP traffic has been sent at different speeds (1200KBps, 10000KBps, 11000KBps and 12000KBps). NetScanPro, Packet Flooder and WinPcap tools were used to send IP traffic at different speeds through the network to hosts. The following rule below tells Snort to prevent any IP packets from any sources to any destinations address from and to any ports.

Drop ip any any ->any any (msg:"Prevent IP traffic"; sid: 100005).

Table 4. 15: Snort NIP-mode reaction to prevent IP traffic.

Speed limits Kilobyte per second	Number Packets analysed of packets received	Total Eth Packets received of the total packets analysed	Total Ip4 analysed Of the Eth Packets received	ICMP packets analysed	TCP packets analysed	UDP packets analysed	Number of IP packets Dropped	% of packets prevented
1200-KBps	232867	100.00%	99.988%	21	0	232817	232838	100.00%
10000-kBps	266764	100.00%	99.991%	13	0	239249	239262	100.00%
11000-kBps	306188	100.00%	99.986%	41	0	231077	231118	100.00%
12000-kBps	351467	100.00%	99.991%	47	0	205152	205199	100.00%

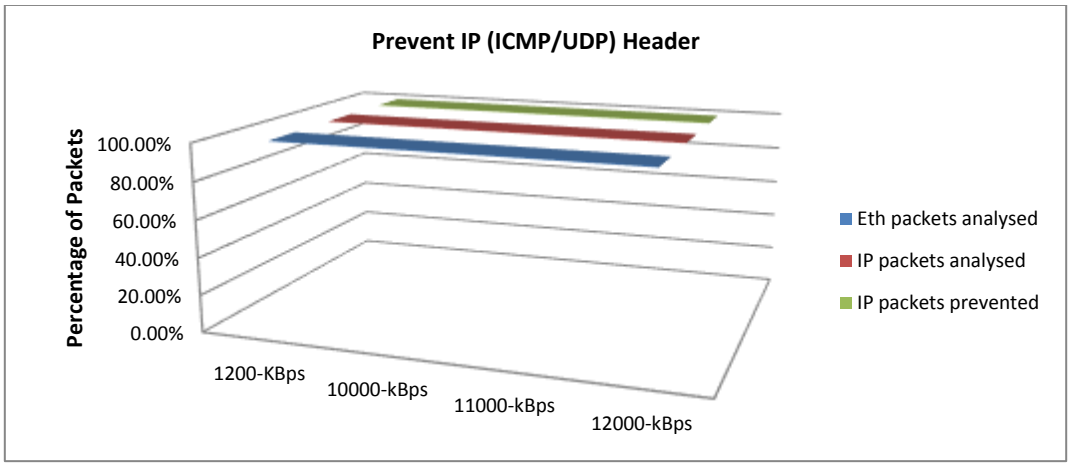


Figure 4. 15: Snort reaction to prevent IP header in high-speed traffic.

As the results in Figure 4.15 show, Snort analysed all packets that reached the system. The experiment shows that Snort NIDPS prevents all unwanted traffic event if it came in high-speed (over 12000KBps).

4.5.2 Experiment 10: Snort NIDPs reaction to block (prevent) TCP header

TCP/IP packets have been sent at different speeds ((1200KBps, 10000KBps, 11000KBps and 12000KBps) by using NetScanPro, WinPcap and Tcpreply tools. The following rule was used to prevent any TCP packets from any sources to any destinations address from and to any source and destinations ports.

Block tcp any any ->any any (msg:"Prevent tcp packets"; sid: 100006 ;)

Table 4. 16: Snort NIP-mode reaction to prevent tcp traffic.

Speed limits Kilobyte per second	Number packets analysed of packets received	Total Eth Packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	TCP Packets block	% packets prevent
1200-KBps	222821	100.00%	19	222786	16	222786	100.00%
10000-kBps	236652	100.00%	23	236615	14	236615	100.00%
11000-kBps	296258	100.00%	45	15054	28	15054	100.00%
12000-kBps	301456	100.00%	80	10152	30	10152	100.00%

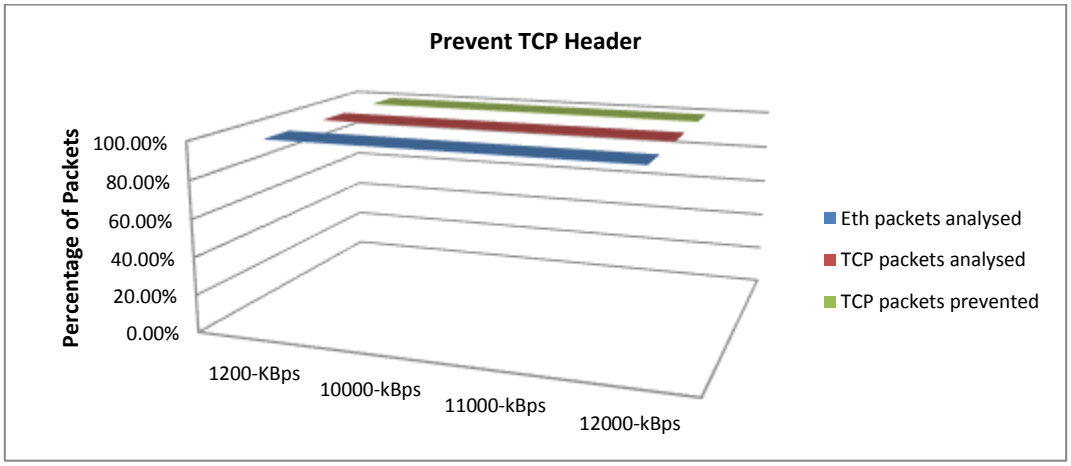


Figure 4. 16: Snort reaction to prevent TCP header in high-speed traffic.

Figure 4.16 and Table 4.16 show that Snort prevented all TCP packets even if the traffic came at high-speed (over 12000KBps).

4.5.3 Experiment 11: Snort NIDPS's prevention mode reaction to reject (prevent) malicious packets

Flood packets and WinPcap were used to send malicious traffic at varying transmission speeds to specific network and hosts. The following rule sets Snort to prevent any UDP packets which contain content '.H`..OK.' and which have TTL of 128 and that travel from any source and ports to any destination and ports.

reject udp any any ->any any (msg: "Prevent Malicious UDP Packets"; ttl: 128; content:|' C2 48 60 AE 97 4F 4B C3 '|; Sid: 100007 ;)

In this experiment, flood traffic was sent at different bandwidths (speeds) (see Table 4.17) with 255 malicious UDP packets (threads) in interval packets with a delay of 1 microsecond (1 mSec). Snort was set to detect UDP threads by using two rule conditions (TTL and content). Use of these

options will prevent any UDP malicious packet that is matched with the TTL value equal to 128 and a data pattern inside the malicious packet with content ".H`..OK.". The hexadecimal number ('C2, 48, 60, AE, 97, 4F, 4B, C3'), which the rule contained, is equal to the ASCII characters ('., H, `, ., ., O, K, . ').

Table 4. 17: Snort NIP-mode reaction to prevent malicious packets.

flood traffic (BytePerSeconds) With 255 UDP malicious packets in (1mSec)	Number Packets analysed of packets received	Total Eth Packets received of the total packets analysed	Total Ip4 Analysed of the Eth Packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	malicious packets reject	% packets prevent
100-Bps	267032	100.00%	89.066%	28	0	237777	237777	100.00%
1000-Bps	266863	100.00%	99.991%	7	0	235338	235338	100.00%
10000-Bps	329926	100.00%	99.988%	522	0	15092	7585	50.258%
60000-Bps	335143	100.00%	99.992%	784	0	186811	32812	17.564%

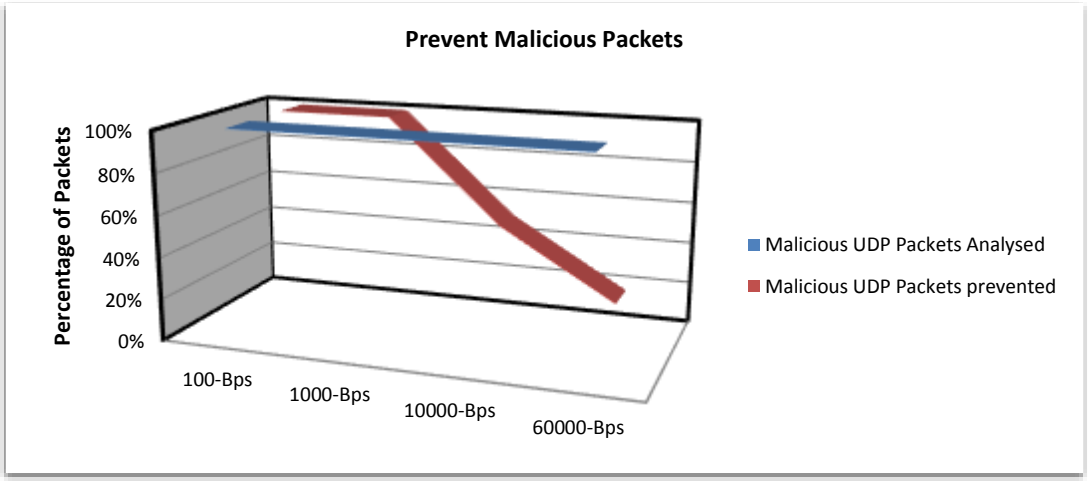


Figure 4. 17: Snort reaction to prevent malicious packets in high-speed traffic.

As shown in Figure 4.17, Snort analysed every packet that reached the wire. When malicious UDP packets were sent at a speed of 1 mSec and flood traffic at 100 bytes per second (Bps), Snort prevented 100% of the total UDP packets that it analysed. As the flood traffic (speed) was increased to 10000 bytes per second (Bps), Snort prevented less than 51% of the total malicious packets analysed (see Table 4.17).

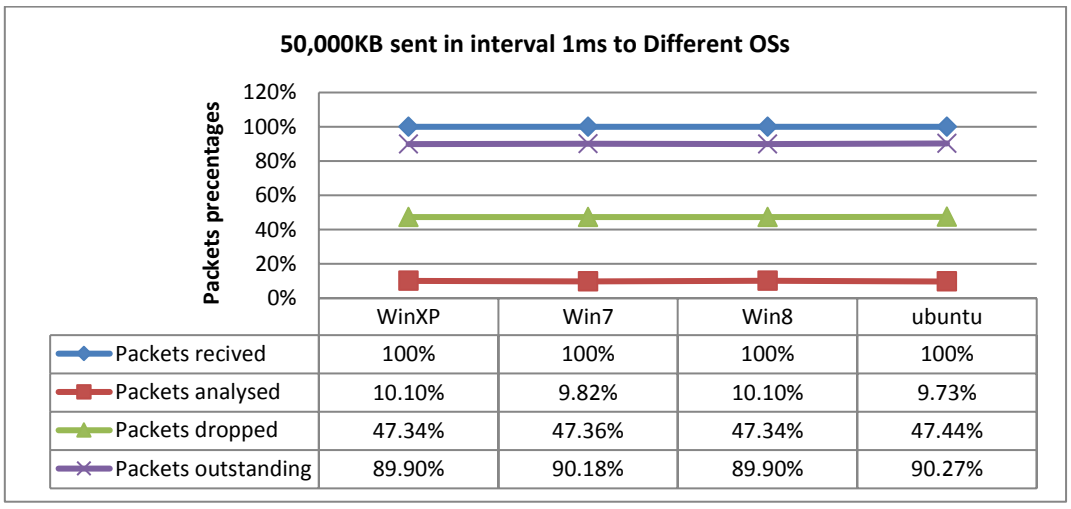
Figure 4.17 shows that the number of missed malicious packets increased when the speed increased. The experiment shows that, when the speed was 60000 Bps, Snort only prevented just nearly 16% of 100% of the malicious packets analysed (see Table 4.17).

4.6 Experiment 12: Snort NIDPS performance under different OSs, buffer size and processor speed.

This experiment was designed to show how the performance of NIDPS is different under different operating systems (OSs), processor speed and different buffer sizes (speeds). Three tests were developed. The experiments were carried out in a virtual environment (see section 3.9.2) and are listed in Table 4.18. In each experiment more than 50,000 packets were sent at an interval delay of 1ms. The size of each packet was 1KB.

Table 4. 18: Experiments for different OSs, processor speed and different buffer sizes.

Test Number	Purpose	Result
1	Evaluate NIDPS performance under different OSs	Little difference between OSs
2	Evaluate NIDPS performance under different processor speeds.	Performance improvement with increased processor speed
3	Evaluate NIDPS performance under different NIC buffer speeds.	Performance improvement with increased NIC buffer speed



	WinXP	Win7	Win8	ubuntu
Packets recived	100%	100%	100%	100%
Packets analysed	10.10%	9.82%	10.10%	9.73%
Packets dropped	47.34%	47.36%	47.34%	47.44%
Packets outstanding	89.90%	90.18%	89.90%	90.27%

Figure 4. 18: NIDPS performance for different OSs processors.

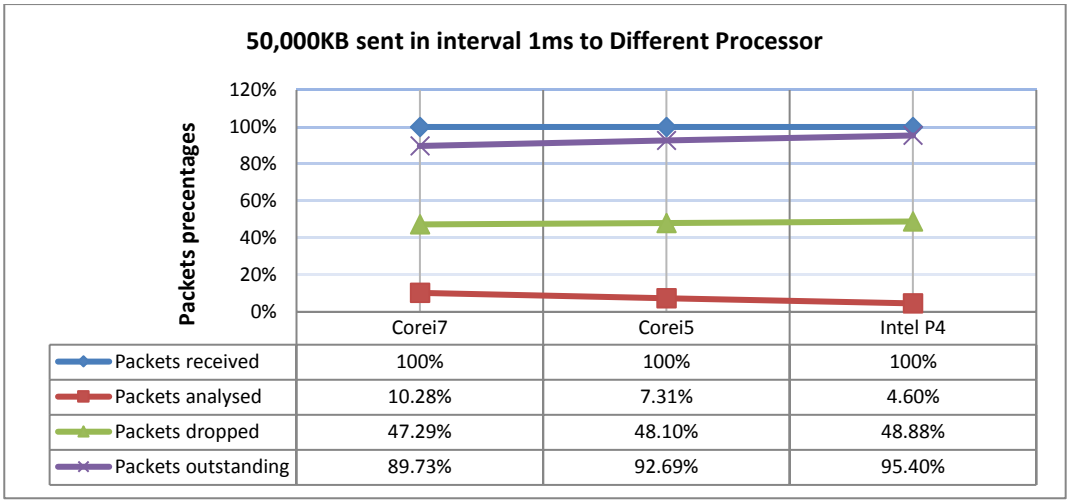


Figure 4. 19: NIDPS performance for different processors speeds.

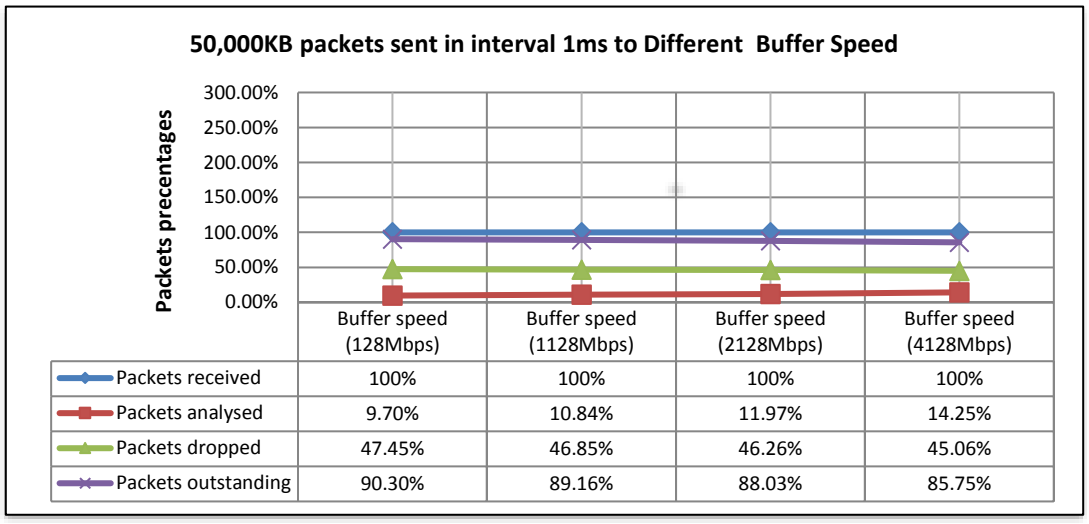


Figure 4. 20: NIDPS performance for different NIC buffer speeds.

As shown in Figures 4.18, 4.19 and 4.20, when the system was tested under different OSs, such as Win 8, 7, XP and UNIX with the same processor and buffer speed, there were no differences in the number of packets dropped (see Figure 4.18). On the contrary, when we tested Snort with different processor speeds (Intel Pentium® D CPU 2.2GHz, Intel® corei5 2.27GHz and Intel® corei7 2.40GHz) and also different buffer speeds (size), the differences between them were considerable (see Figures 4.19 and 4.20). As shown in Figure 4.19, Corei7 dropped fewer packets than the others. Figure 4.20 shows that while buffer speed increases, fewer packets are dropped and outstanding, and more packets will be analysed. These experiments show that the important components in NIDPS performance are buffer size and processor speed.

4.7 Summary of experiments

Experiments 1 to 4 have showed that Snort drops and outstanding packets in high-speed network traffic. They show that Snort NIDP's performance analysis was affected when it was deployed in high-speed and high-load traffic. Experiment 5 to 8 showed how Snort-NIDPS's performance detection decreased while traffic speed limit on the network is increased. They show that Snort NIDPS missed malicious packet alerts when it is implemented in a speedy network. Experiments 9, 10 and 11 showed Snort cannot prevent all unwanted traffic (malicious packets) when the traffic comes at high-speed. The experiments show that Snort NIDPS performance prevention will be at a lower performance when traffic speed is increased. Finally, experiment 12 shows that the limitation of buffer size and processor speed affect Snort NIDPS performance.

4.8 Conclusion

This chapter has reported on experiments carried out to test NIDPS performance under high-speed and high-volume traffic. Snort NIDPS performance has been tested under different circumstances (speed and volume). The results show that Snort NIDPS has been affected and is unable to analysis traffic and control unwanted packets when faced with high-speed traffic.

CHAPTER 5: PROPOSED SOLUTION

5.1 Introduction

The results of the experiments described in chapter 4 shows that the NIDPS's performance decreases when faced with heavy and high-speed attacks. This chapter outlines a novel solution that uses QoS configuration and parallel technologies in a layer 3 switch in order to increase NIDPS performance in the analysis, detection, and prevention of malicious attacks. The chapter describes the technical problems that affect NIDPS performance and provides a novel solution to the problems of dropped and outstanding packets, of lost alerts and logged packets, and of inability to prevent (blocked) unwanted packets. These can be a prevalent issue for NIDPS performance in heavy and high-speed traffic environments.

5.2 Proposed Solution

Critical analyses were done for the previous experiments presented in chapter 4 (see Figures 4.1 to 4.10, respectively). The figures show that performance of Snort's throughput is affected when Snort NIDPS is exposed to a high-volume and speed of traffic; more packets will be dropped and left outstanding as the size of packets and the speed of traffic increases. Figures 4.11, 4.12, and 4.13 show that Snort's performance detection decreased when the traffic speed increased. There were more missed alerts and missed logs for packets as the speed of traffic increased. Furthermore, when the malicious traffic was sent at high-speed, Snort lost more malicious packet alerts and logs (see Figure 4.14). Figure 4.17 shows that Snort NIDPS cannot prevent unwanted and malicious packets in a high-speed traffic environment. Furthermore, when the Snort performance was tested under different OSs, buffer size and processor speed (see Figures 4.18, 4.19, and 4.20), the experimental results show that Snort performance was positively increased while the processor and buffer speeds were increased.

According to the results shown in the previous chapter, the following activities affect Snort-NIDPS performance:

- high-speed traffic
- large sized packets
- high-volume of traffic

However, Snort is used as a real-time traffic processor on the network. It is a multimode packet tool that can perform network traffic analyses, intrusion detection or prevention, and content

searching/matching in real-time as well as for forensic post-processing. Snort has a limited time in which to analyse traffic and then alert, log, and prevent any unwanted and malicious packets; it will drop or leave outstanding packets without analysis if the speed of a network's traffic is higher than Snort's processing limit. Snort will also miss detection and prevention of unwanted traffic in this circumstance. The performance of an NIDPS could be described as ineffective if the NIDPS is unable to detect or stop unwanted packets that could reach the system. Based on literature review in chapter 2 and experiment 12 in chapter 4, there are two main causes of ineffective NIDPS: buffer size and processing speed.

When traffic moves through the network interface card (NIC) to the NIDPS node, the packets are stored on the buffer until the other relevant packets have completed transmission to processing nodes. In the event of high-speed and heavy traffic in multiple directions, the buffer will fill up. Then packets may be dropped or left outstanding (Kishore, Hendel, and Kalkunte 2015, Naouri and Perlman 2015 and Zhu et al. 2015). In this case, there is no security concern about the packets dropped; the packets are dropped outside the system. The outstanding packets that are waiting or have not been processed by the NIDPS node may affect the system, however.

Packets can also be lost in a host-based IDPS. Most software tools use a computer program such as the kernel, which manages input/output (I/O) requests from software and decodes the requests into instructions to direct the CPU's data processing. When traffic moves from the interface (NIC) through the kernel's buffer to the processor space, where most of processing nodes are executed, the packets will be held in the kernel buffer before being processed by the CPU. When some nodes experience a high-volume of data, the buffer will fill up and packets may be dropped.

Configuring the kernel parameter can enhance kernel performance by increasing the level of optimization and selecting multivariate features such as kernel complex quantitative near-infrared (K-NIR), kernel support vector regression (k-SVR), or kernel partial least squares (K-PLS) to improve the accuracy of packet processing (Salah and Kahtani 2009, Wu, Cadambi, and Chakradhar 2015, Fraser et al. 2015, Lee et al. 2015 and Lutz, Fensch, and Cole 2015). In order to hold and process packets quickly, these kernel performance enhancements pull a high value of packets from interfaces and bind them with obtainable CPU cycles, which limit packet speed and time and have no buffer memory. Furthermore, it requires a great deal of CPU to process a vast amount of data buffered in the kernel; the CPU cycles may run out of time. In this case, the packets that were dropped in the kernel and NIC might drop very early in the CPU cycles, which cannot buffer packets (Smith et al. 2014 and Emmerich et al. 2015). In the cases of network, host, and processor packet loss, NIDPS is affected because packets are dropped before reaching the NIDPS node or dropped after reaching the node but before matching packets with the signatures database.

This research does not focus much on the network-based packets drop, because these are dropped out of the system and lost before they reach the NIDPS. Packets dropped after having been received by the NIDPS are of greater security concern, and thus are the focus of this research. Such packets might reach the target systems when they are missed by the NIDPS. This study also focuses on outstanding packets based in the NIC network. In order to solve the problem of dropped packets in NIDPS, the researcher investigated the use of a QoS configuration in layer-3 switches with parallel NIDPS technology to organise and improve the interval packets processing speed. The increased speed should improve throughput performance of NIDPS, even during a high-speed, high-volume traffic environment.

One mechanism that QoS offers is queue technology, which can give a switch a new logical throughput-traffic-forwarding plan. A configurable QoS offers two input queues and four output queues at the physical switch interfaces, which might be switch virtual interfaces (SVIs) or ports. As Figure 5.1 shows, the switch interface has been configured to have two input queues and four output queues. The queues parameters were configured to allow queues to process traffic as a group of bytes. These load a set of packets equally among the queues and divide traffic into parallel streams in order to increase the rate of packet processing. The system then uses parallel NIDPS to increase the NIDPS throughput performance and analyses each portion of traffic separately to determine whether it is free of malicious codes.

A class map and a policy map were made for each input queue. The class map recognises and classifies a certain type of traffic for each input queue, while the policy map controls and organises the speed limit for each input queue and applies the limit to all interfaces. The bandwidth, threshold, buffer, memory reservation, and priority (queue and traffic) were configured for all ingress and egress queues to treat and control traffic in order to help prevent congestion or complete failure through overload.

To guarantee the bandwidth for an interface including ingress and egress queues, one of two functions can be used inside the switch: Shaped or Share Round Robin (SRR). Network devices such as switches and routers can classify all traffic that flows through. The Shaped task only exists on output queues. With it, a queue books a percentage of a total port's bandwidth and this is guaranteed to that queue and cannot be shared with other queues The Share function (SRR) operates on both input and output queues. It guarantees a queue a portion of a total port's bandwidth, but this bandwidth can be shared with other queues if it is unused. Figure 5.1 shows the architecture of the system. The Share queue has been configured for two input queues to use the maximam efficiency of queue capacity and help prevent congestion each output queue has an individual configuration using

the shaped queue to control and organise traffic speed for each output queuewhich is processed separately via parallel NIDPS nodes.

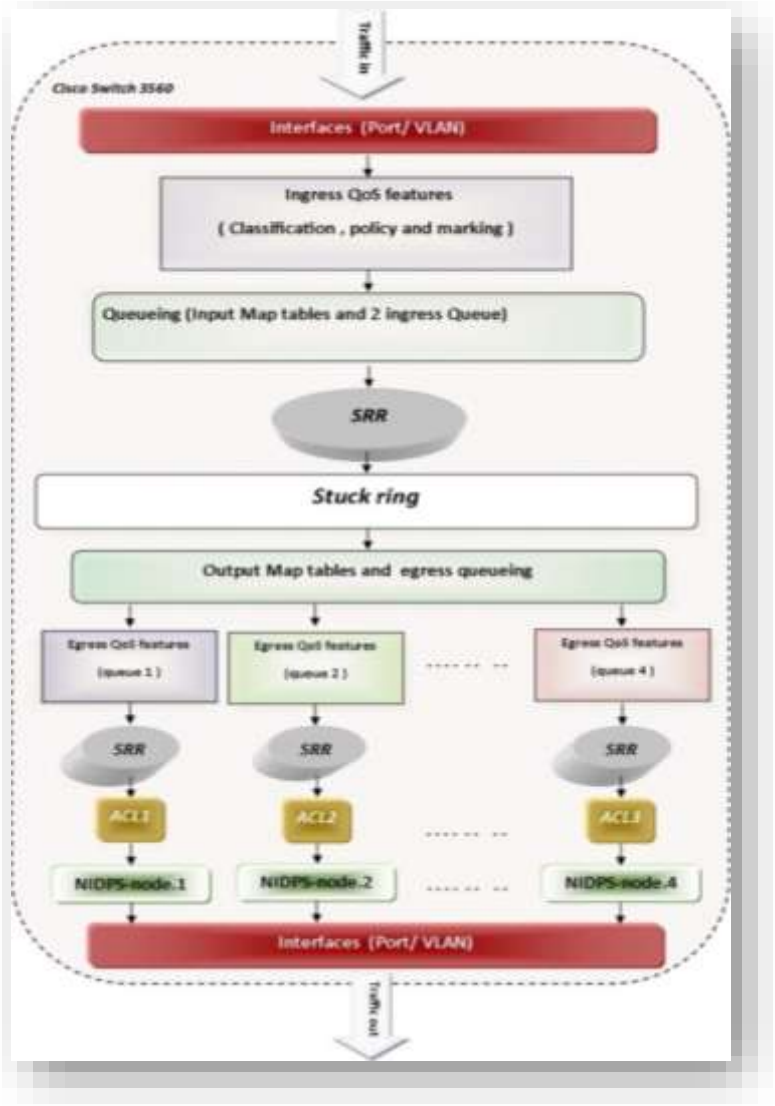


Figure 5. 1: Novel architecture for NIDPS.

One queue was configured as an expedited queue. It prioritised QoS services and did not service other queues until the bandwidth of prioritised queue reaches its limit. A memory buffer reservation technique was set on the proposed novel configuration for each queue to guarantee that each queue's buffer could attain more space once it reached its limit by reserving space from an available queue buffer, from SVI or port interfaces' memory buffers, or by switching to a common memory pool buffer. However, headers such as ICMP, TCP, and UDP as well as malicious packets have different characteristics and techniques. The SRR, threshold, and priority methods for each output queue and ACLs offer a wide range of opportunities to deal with the behaviours of different IP

headers and malicious packets. For example, when all input and output queue buffers are flooded with traffic, priority queue and threshold map values can deny buffer overflow.

The main idea of this novel configuration design is to manage and allocate a specific traffic weight, or set of bytes, into each input queue and process each output queue individually in parallel, thereby increasing NIDPS processor speed and reducing traffic congestion, even if the traffic is high-load and high-speed.

5.3 Technical Discussion of QoS and Parallel NIDPS Configuration

A network intrusion detection and prevention system (NIDPS) is contingent upon the nodes being able to see the traffic on the network between source and destination targets. Traffic identification is frequently prepared at the network border by employing the node on a mirrored port arranged to send the node a copy of the entire packet flow in and out of the network. When some of these packets fail to reach the NIDPS node for analysis, packet drop may occur. The NIDPS node drops/loses the packets because it cannot see them or packets (traffic) bandwidth is over its limit. Two (2) types of packet loss (drop) may occur: packets are dropped before reaching a system; and packets are dropped after reaching a system.

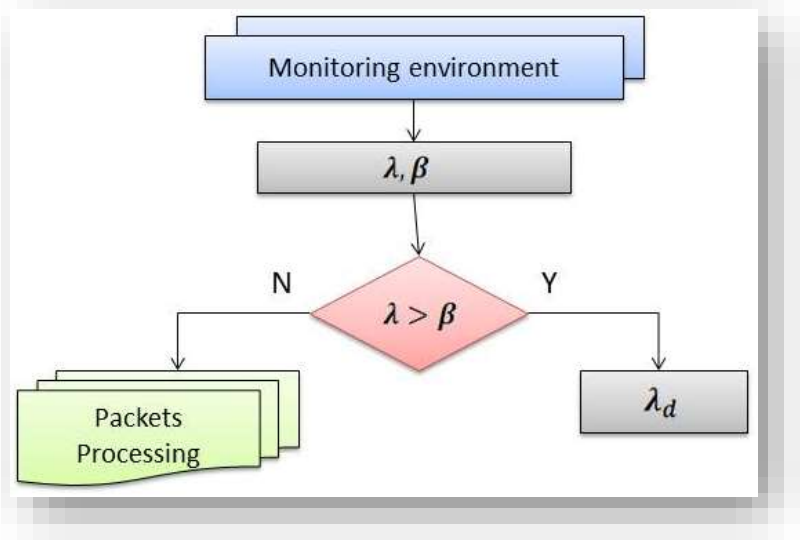


Figure 5. 2: General model of buffer packets drop.

However, there are three (3) places that packets could be dropped: in the network, in the host or in the processor, because all of them are dependent on buffer size and processing speed. If the arrival packet speed rate (λ) is greater than the network or host buffer speed rate (β), dropped ($\lambda_d >$

0) packets may occur (see Figure 5.2), and even increasing the buffer speed can affect processor speed and cause packet drop.

Dropped (λ_d) and outstanding packet (λ_o) rates range from > 0% to nearly 100% of n packets dependent on the traffic arrival speed and traffic load. Various traffic is applied at different speeds (λ) and is organised through the configuration in datasets of bytes. The resulting abridged datasets are analysed at the NIDPS node to show analysed packet rates and lost packets rates (dropped or outstanding packets rates).

For network based packet loss, the NIDPS node fails to analyse this traffic (packets) because the network drops packets and the node cannot see them. Packet loss has no negative impact on the node's ability to detect or prevent internal malicious packets, but it does have an impact on the receiving system in that useful packets would not be delivered. In host based and processor based packet loss, the NIDPS node has analysed this traffic because these packets have reached the host system but the NIDPS node has not been able to process them. This kind of packet loss has a negative impact on the node's ability to detect or prevent attacks. In this research, a novel design based on a Layer 3 network switch was proposed to reduce this kind of packet loss.

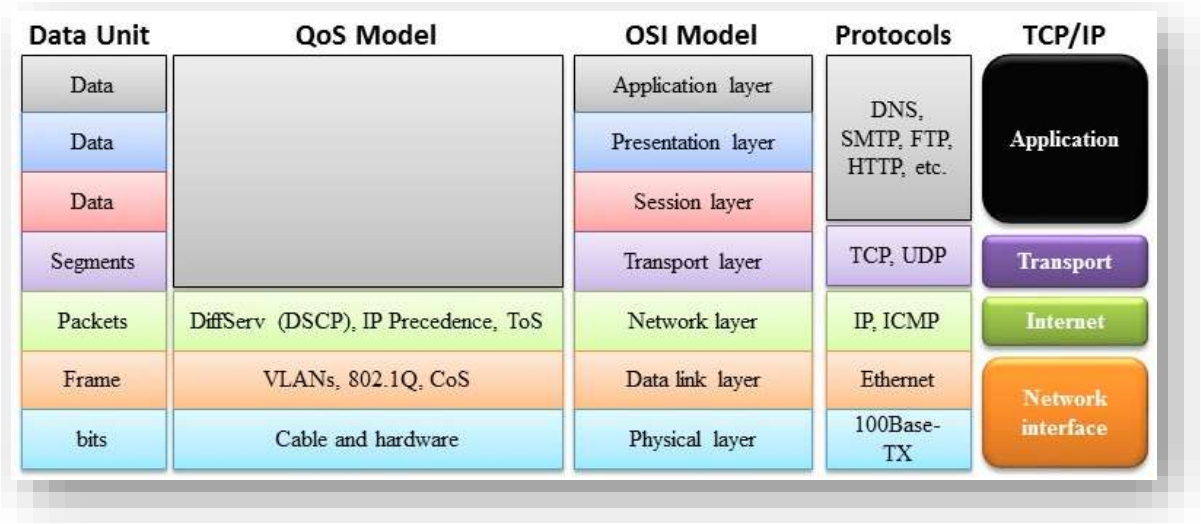


Figure 5. 3: Positioning of CoS and DSCP values.

A layer-3 switch enables a network to get the best performance effort from a network traffic delivery system. Through it, packets of various priorities can be delivered on network in a timely manner. When networks experience high-speed and heavy traffic, each packet has a similar chance of being dropped or modified. Implementing QoS methods, such as queueing, memory reservation, congestion-management, and congestion-avoidance techniques, can yield preferential treatment to

prioritise traffic according to its relative importance. Furthermore, QoS technology ensures that network performance is more predictable and that bandwidth utilization is more effective (Cisco 2014a:42). QoS can be configured on physical interfaces such as ports and SVIs (Szigeti et al. 2013:294-299 and Cisco 2016a:826).

By default, most of the switches work in layer 2, which is the data link layer. The switches use the class of services (CoS) value (see Figure 5.3), which enables differentiation of the packets (Szigeti et al. 2013 and Cisco 2016a:827-828). However Layer 2 provides insufficient methods to support switch features such as QoS features, dynamic access control lists (ACLs), VLAN features, static IP routing, and policy-based routing (PBR) Cisco-default Smartports (Cisco 2014b, Bul'ajoul, James and Pannu 2015 and Cisco 2016a).

Other mechanisms operate at Layer 3 (see Figure 5.3). For example, differentiated services (DiffServ) allow different types of services to be offered depending on a code (configuration). DiffServ allows a policy that gives priority to a certain type of package (Szigeti et al. 2013, Cisco 2014a:42 and Cisco 2016a:827-828). DiffServ architecture is the basis for the QoS implementation in this research. It assigns each packet a classification upon entry that states its priority and its likelihood of being delivered into a network before packets are distributed. It adjusts each packet for different traffic speeds to ensure timely delivery.

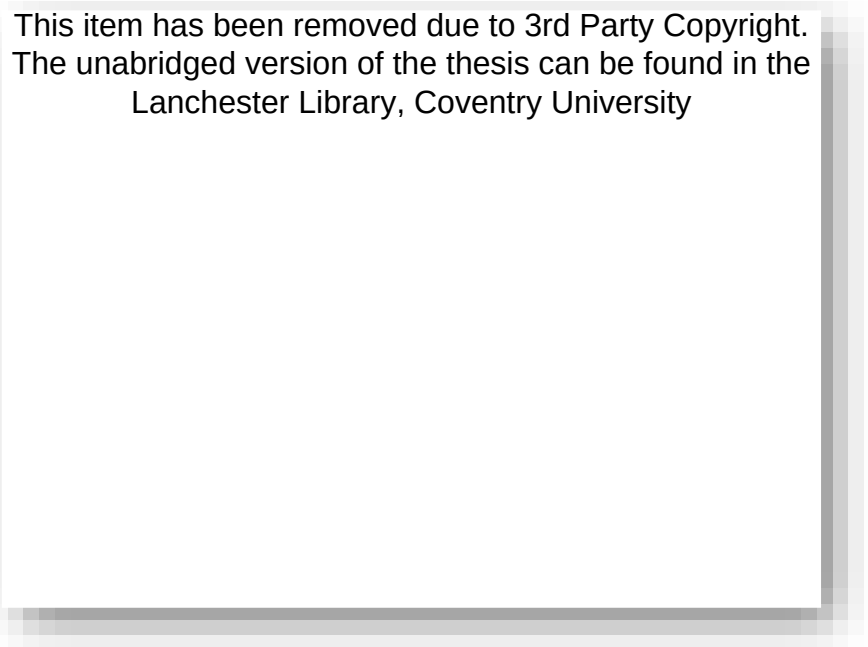


Figure 5. 4: QoS classification bits in frames and packets (Cisco 2014a:11and Cisco 2016a:828).

Figures 5.3 and 5.4 illustrate the relative layers at which CoS and Differentiated Services Code Point (DSCP) operate. QoS has supported the use of both values because DSCP values are backward-compatible with IP precedence values. Layer-3 DSCP values range from 0 to 63, while IP precedence values range from 0 to 7 (Szigeti et al. 2013:191, Cisco 2016a:827:830-833).

The static and dynamic classification methods involved Layer 3 header information matching. The IP precedence or DSCP value indicated the type of service required. For example, a dynamic classification access list can be used to identify what IP traffic can be placed into a reserved queue.

Classification can also take place in the Layer 2 frame. Packet classification should occur close to the edge of the network because it is processor-intensive and every hop must decide how a packet should be treated. Setting the type of service (ToS) field in the IP header can be used to achieve a simpler classification which can be carried with the packet across the network (Szigeti et al. 2013:32 and Cisco 2016a:830-833). In Layer 2, 802.1Q and 802.1p frames use 3 bits for the IP ToS field; in Layer 3 IPV4 packets use 6 bits for DSCP in the ToS field to carry the classification information (see Figure 5.4). Regardless of a network's capability to identify and classify IP packets, hops can offer each IP packet a QoS service. Special techniques can be configured to approve a priority queue in order to ensure that a large data packet does not interfere with transmission.

"If a node can set the IP Precedence or DSCP bits in the ToS field of the IP header as soon as it identifies traffic as being IP traffic, then all of the other nodes in the network can be classified based on these bits. In most marking of IP networks, IP Precedence or DSCP should be sufficient to identify traffic as IP traffic" (Szigeti et al. 2013:191-247 and Cisco 2016a).

In the configuration method utilized by this research, the first action changes the switch frame from Layer 2 to Layer 3 by mapping values from CoS to DSCP (See Appendix1:Section 1. Part 1). Because differentiated services technology can offer more precise handling of traffic on the network, can classify each packet upon entry into the network interfaces, and allows for adjustments to be made for different traffic speeds and loads, the researcher considered DSCP to be the best choice for the intended usage. The mapping action between values determines the delay priority of the packets. CoS has 8 values and DSCP has 64 values. Thus, the DSCP values allow for a higher degree of differentiation (Szigeti et al. 2013 and Bul'ajoul and Pannu 2015).

QoS features such as a policy map and class map can be given preliminary management to classify the traffic inside the switch with the same policy and class plan; different management can be given to packets with different class and policy plans (Szigeti et al. 2013:32 and Cisco 2016a:833). Classification is the process of identifying the data packets to a class or group in order to manage the

packet appropriately (Szigeti et al. 2013a). Network devices use several match criteria to place packets into classes that can be intensively processed if nodes must repeat classifications based on access list matches. The class information can be assigned by switch, router, or end host. Therefore, the node will mark packets as soon as they are identified and classified. Policing involves creating a policy that defines a group weight for the traffic (the number of bytes to be processed together) and applies it to the interface. Policing can be applied to a packet per direction and can occur on the ingress and egress interfaces. Different types of traffic can be recognised in terms of type and ports, and differentiated policies can be set accordingly.

Network QoS technology enables the implementation of a new logical and throughput-traffic-forwarding plan in the switch. For the purpose of this research, a physical interface was configured to two input queues and four output queues (See Appendix 1: Section 1. Part 1). This configuration helps to prevent congestion traffic (which would cause buffer overflow) and helps to improve buffer throughput performance. A buffer was set for each queue and a memory reservation method using a dynamic memory reservation technology was created and implemented in order to organise and hold more traffic. After all packets were placed into input queues, class and policy maps were implemented to packets based on their QoS requirements (See Appendix 1: Section 1. Part 3). Appropriate services were then provided, including bandwidth guarantees, thresholds, queue setting, and priority servicing through an intelligent ingress and egress queueing mechanism (See Appendix 1: Section 1. Part 3, 4 and 5).

The packets' class map information is assigned along the path of a switch. QoS users can use this information to limit the volume of incoming packets distributed to each traffic class. The default behaviour in Layer 3 switches handle packets using the DiffServ architecture using "per-hop" behaviour (Szigeti et al. 2013:6:940-941 and Cisco 2016a:828,). If a switch along the path does not provide a consistent behaviour per hop, QoS provides a conceptual and constructed solution, such as an end-to-end queue solution. The solution is based on a configurable policy map that allows the system to examine packet information closer to the edge of the network, which helps prevent the core switch from experiencing overload. The output queues are processed individually where parallel Snort NIDPS nodes are implemented (each output queue has own NIDPS node) (see Figure 5.5) in order to enhance the performance of NIDPS and increase packet processing speed.

Queues and class and policy map technologies can use access control lists (ACLs) to allow the processing management of different types and patterns of incoming and outgoing packets (Cisco 2014b:1). The novel configuration proposed in this research uses an ACL technology with a class map and SVI queues, as well as a policy map that specifies each type of IP traffic to be processed by

implementing parallel output queues with associated parallel NIDPS nodes (see Figure 5.6) (See Appendix 1: Section 1. Part 2).

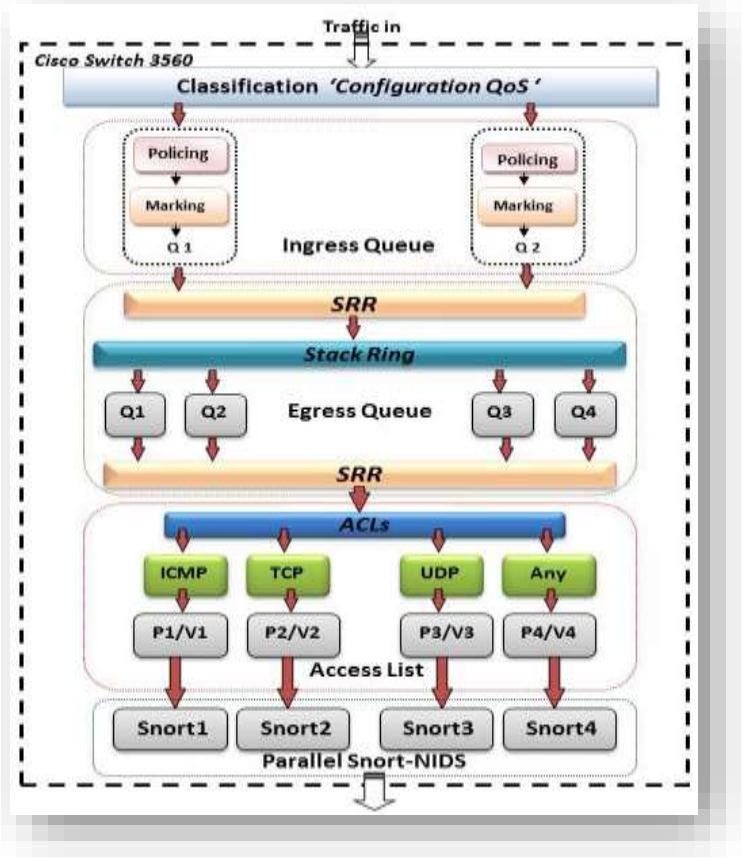


Figure 5. 5: Parallel Snort-NIDPS Using QoS and ACLs.

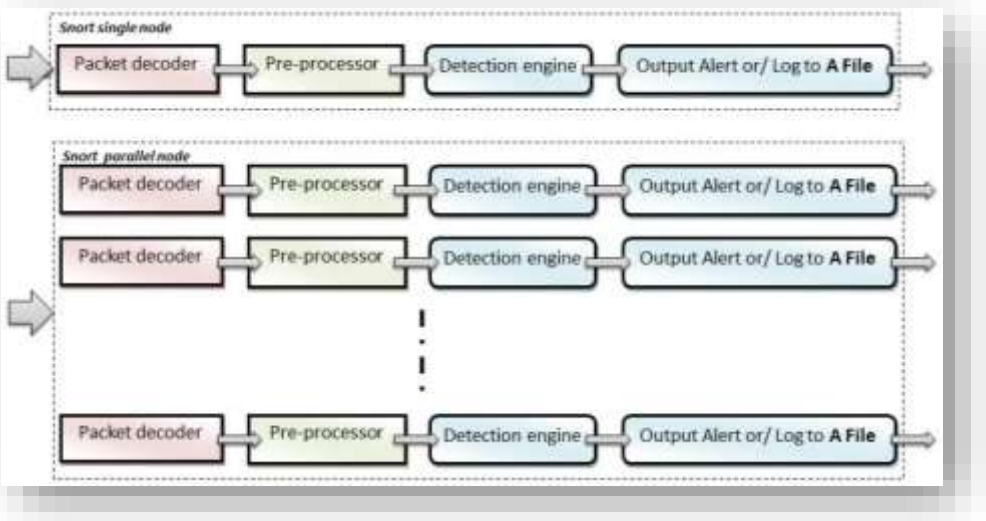


Figure 5. 6: Snort NIDPS parallel node.

This novel QoS configuration includes a novel architecture of packets classification as well (see Figure 5.7). Data were classified through a class map that which enabled packets to be processed as a group of bytes (See Appendix 1 :Section 1. Part 2) defined by a policy and ACLs that were matched with DSCP values. A policy map was made to specify what traffic classes should inspire action. The following actions constitute the method:

- Classify the ACL traffic with a class map for SVI and ports
- Organise a rate-limit for each group of bytes for the class traffic
- Establish the matching of DSCP values with classes
- Set a particular DSCP value to be mapped with classes

After packets were classified and policed with a specific bandwidth, some were dropped out of the profile. Each policy can specify what actions should be carried out (Szigeti et al. 2013 and Cisco 2016a:833), including dropping packets, allowing dropped packets to be modified, allowing packets to pass through without modification, and deciding on a packet-by-packet basis whether a packet is in or outside the profile. This novel QoS policy map architecture was proposed as follows:

- Packets dropped were modified to be re-processed again and mapped with new DSCP values based on the original QoS label.
- When packets are reprocessed, they may get out of order. To prevent this, a policy was designed to allow packets to be re-processed in the same queue as the original QoS label.
- The system has the ability to mark up a limit speed (as a set of bytes) for each input queue.
- If packets are not matched with DSCP values, packets will be dropped. See Figure 5.7 for an illustration of the architecture (See Appendix 1:Section 1 .Part 2).

A hierarchical policy map was created and applied it to the traffic path inside the ingress physical queues. The policy map targeted SVIs and ports. Two types of QoS policy were created: individual and aggregate. Individual QoS applies a separate policy to specify a bandwidth limit for each traffic class. Aggregate QoS specifies an aggregate policy with which to apply a bandwidth limit to all matched traffic flows. The individual policer only affects packets on a physical port. Furthermore, if more than one type of traffic needs to be classified, it is possible to create more ACLs, class maps, and policy maps (Szigeti et al. 2013). In our experiments, three types of traffic (TCP, UDP, and ICMP) were classified using three criteria groups: ACL, class map, and policy map. Each type of traffic matches an instance of these related criteria groups and enables the matching time of arrival traffic with NIDPS node to be reduced (See Appendix 1 : Section 3).

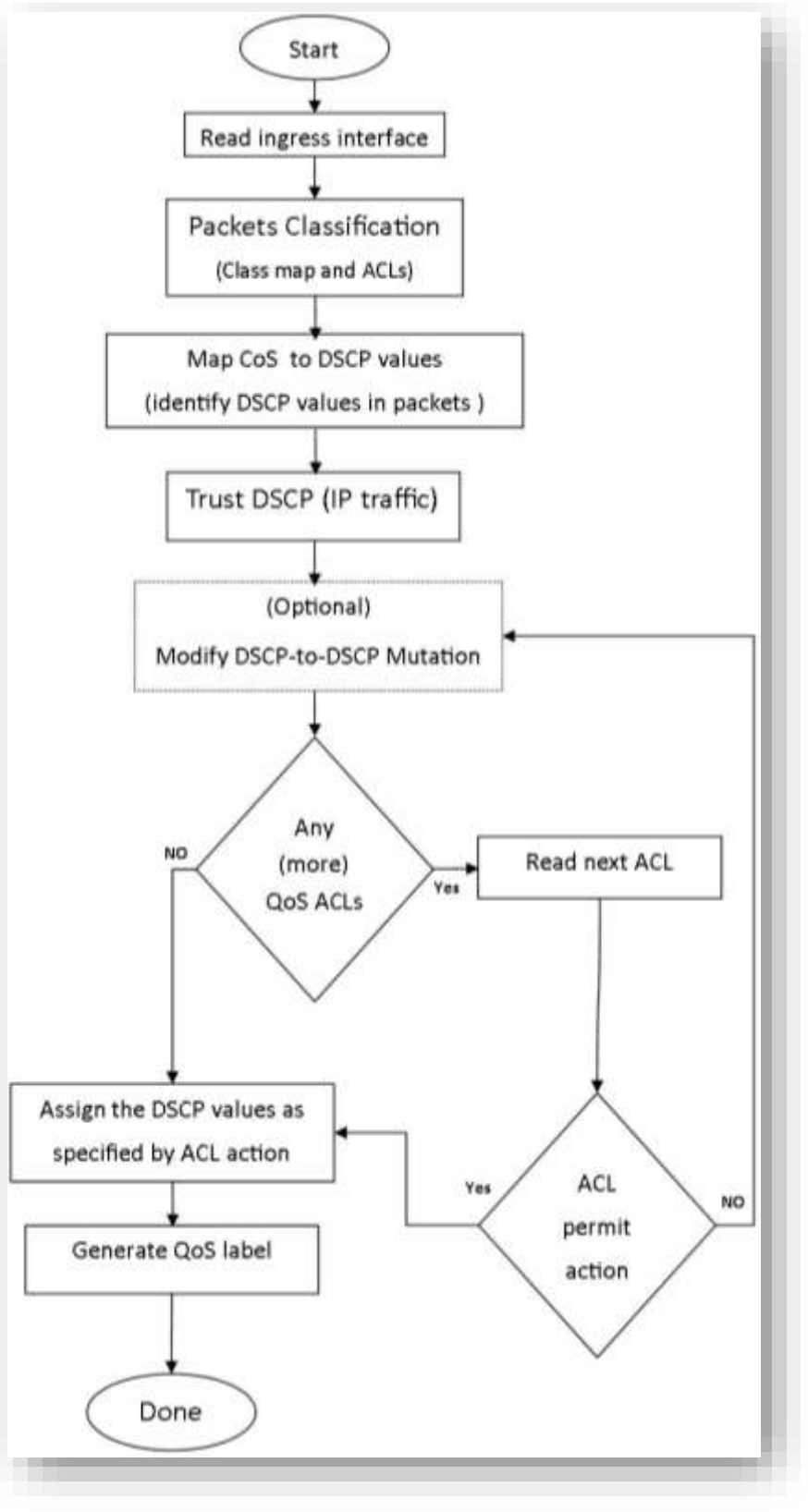


Figure 5. 7: Novel architecture of packet classification and marking.

Switches receive each traffic frame in a token bucket (Szigeti et al. 2013:62 and Cisco 2016a:835), which is defined as an algorithm used in packet switches to check leaks of data transmissions. It is set at the same rate as the configured average packets rate and conforms to defined limits on bandwidth to allow a burst of traffic for short periods. It is used in traffic policy mapping to prevent the problem of the bucket hole overflow. Each time a token is added to the bucket, the algorithm checks to see if enough room is available in the bucket. If not, the packets will be marked as non-conforming, and the specified policy action will be taken. In the novel QoS architecture of this research, packets dropped out of profile were marked down with new DSCP values and the DSCP value was modified to generate a new QoS label (see Figure 5.8) (See Appendix 1:Section 1.Part 2).

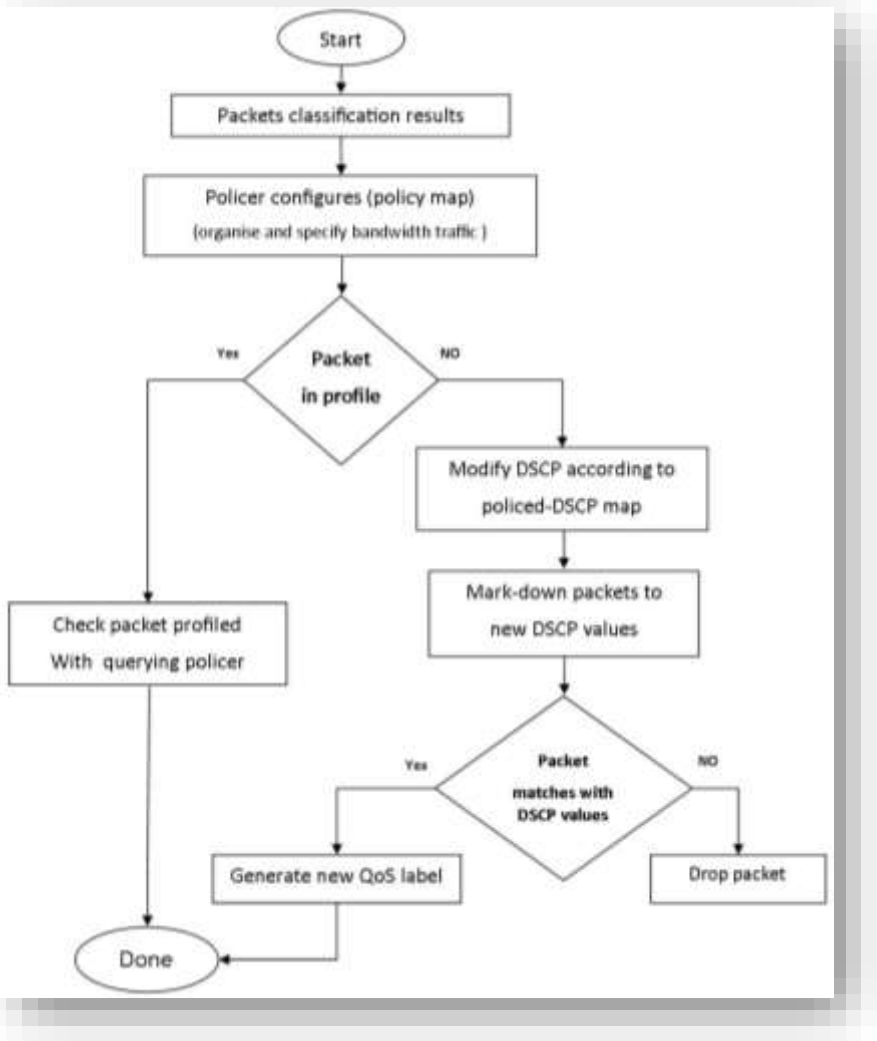


Figure 5. 8: Novel architecture of packet policing and marking.

During traffic classification, QoS employs configurable map tables to match a corresponding DSCP from a received CoS, IP precedence, or DSCP value. If DSCP values are different between

QoS domains, a configurable mutation map (DSCP-to-DSCP values) can be used. Throughout QoS policing, the DSCP value is assigned according to the IP traffic. This value assignment creates a policed-DSCP map (Szigeti et al. 2013 and Cisco 2016a).

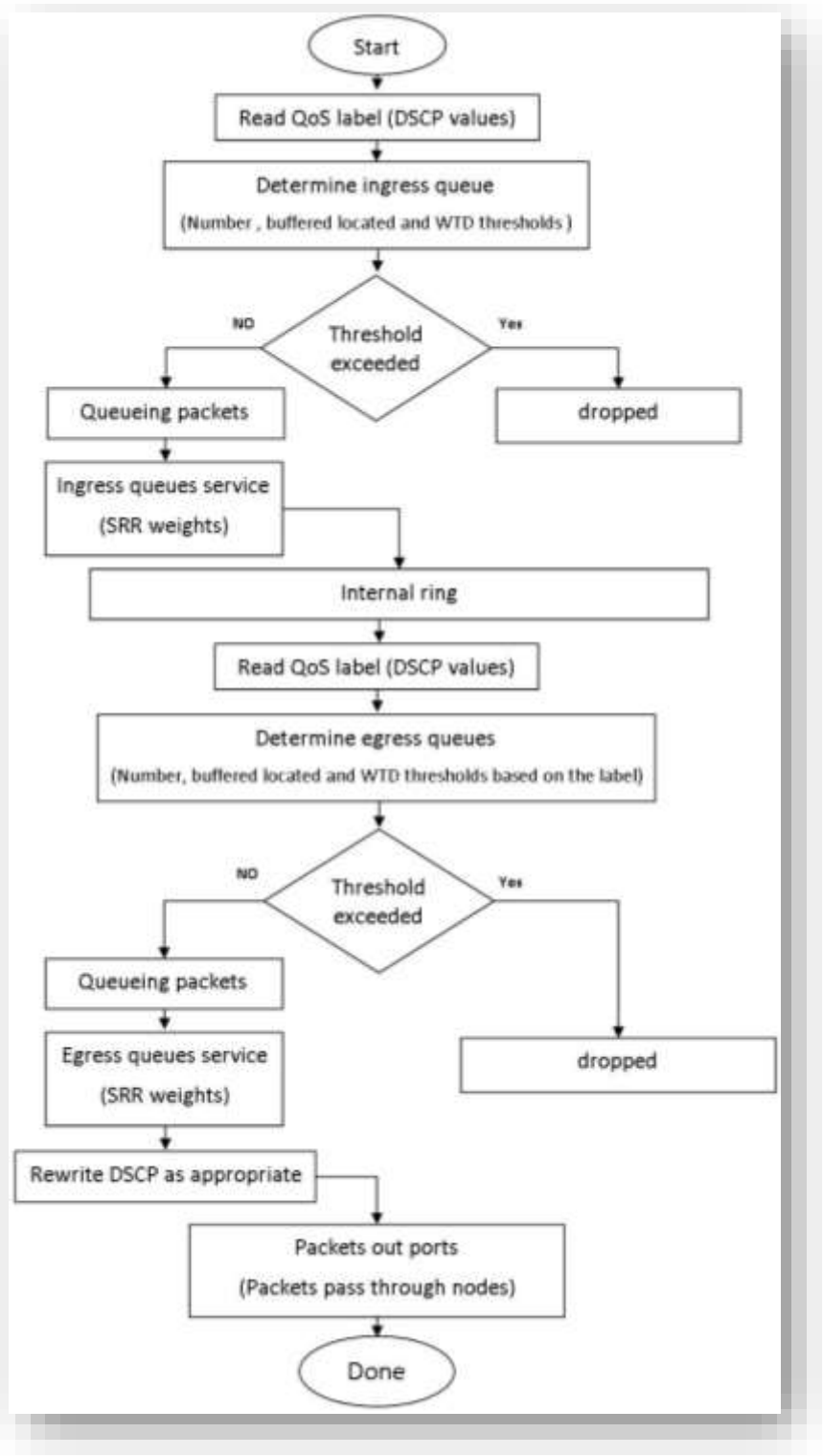


Figure 5. 9: Novel scheduling architecture for ingress and egress queues.

QoS stores packets in input and output queues according to the QoS label, which is defined and identified by the DSCP values in the packets. Threshold map values can be selected through the DSCP ingress and egress values. The QoS label also identifies the weighted tail drop (WTD) threshold value (Szigeti et al. 2013:260 and Cisco 2016a:838). In the novel QoS architecture created in this research, WTDs were employed on ingress and egress queues to cope the bandwidth length of each queue and deliver the drop precedence for different classifications of packets (See Appendix 1:Section 1. Part 2, 4 and 5). As each packet is assigned to a specific ingress and egress queue, WTD allows the packets assigned under the QoS label to be subject to different thresholds. If the available space on a destination queue is less than the volume of packets, the threshold is exceeded for that QoS label and the switch drops the packet (Szigeti et al. 2013:260 and Cisco 2016a:839). In this research (see Figure 5.9), DSCP values were mapped to ingress and egress queues and located buffer space. WTD thresholds for input and output queues were set. Each queue has three drop thresholds. This means that different thresholds can be set for different types of packets. Each value of the threshold represents a percentage of the queue's total buffer. Furthermore, one of the important techniques that used in the QoS architecture is a buffer reservation.

Buffers are universal throughout the software and hardware layers of any network computer system. They are valuable in reducing the impact of traffic rate variability on the network especially in the case of traffic rate points. By having sufficiently large buffers to absorb traffic rate spike, high latencies associated with retransmissions and lost data (traffic) can be avoided. They are also useful if there is a temporary difference in the rate at which traffic is produced and consumed. However, increasing the buffer size cannot compensate for packet processing that is perpetually slower than the packet arrival rate. A switch may have different buffer configurations. The total rate of all buffers is β; and each ingress and egress buffer of an interface is limited to rate α. The same α applies to all interfaces. The rate of a buffer is the speed at which packets move out of it and this depends on the underlying processing system.

The multi-interfaces switch manages buffering across a number of ingress interfaces (ports). An ingress interface has ingress buffers connected to common egress buffers. The switch algorithm is also responsible for scheduling. At each event of scheduling, the switch algorithm selects one of the ingress buffers. The packet at the head of the selected buffer is then transmitted to the inside at the switch and via the egress buffer to the target system. There are some formulations that model the entire switch rather than just one interface (Kawahara, Kobayashi and Maeda 2015). For example: a switch consists of n ingress and n egress interfaces, where each interface has buffer. An arrival event (packets) arrives at the ingress interfaces (which have specified destination egress interfaces) . At the scheduling event, packets at the top end of the ingress interface buffers are sent to the egress interface buffers. Here, the switch algorithm matches the packets in the ingress and egress interfaces.

According to this matching, the packets in the ingress interfaces will be transmitted to the corresponding egress interfaces. In this scheduling task, there is also a buffer for each pair of ingress and egress interfaces. Thus there arises anther buffer management problem at scheduling phases. In the implementation of QoS architecture proposed in this thesis, QoS DiffServ was used to assign a value to each packet according to its importance and then it decides the order of packets to be processed through queues based on the value of packets. Additional buffers were provided dynamically to the ingress and egress interfaces. A QoS switch was used to control the input and output traffic. A priority queue was implemented for one of the ingress queues in an interface.

By default, the ingress buffer rate is the same as egress buffer rate. However, when the arrival event of traffic (packets) rate (λ_{in_i}) is more than total rate of egress buffers, or one of n egress buffers already reached α , the packets would be dropped (λ_d>0). In the novel configuration, Sharing policy was configured for each ingress-queue's buffer which corresponds to rate $\frac{\alpha}{2}$ and the egress-queue's buffer was to rate $\frac{\alpha}{4}$, where α is the maximum rate for the interface's buffer. All buffers were assumed to have the same maximum rate.

WTD (Weighted Tail Drop) is employed on the queues to manage the queue lengths and to be responsible for drop precedence for different packets classification. It used the frame's assigned QoS label to subject it to different thresholds. If the thresholds are exceeded for that QoS label (the available buffer space in the destination queue is less than the size of n packets), the switch drops n packets (Cisco 2016a:839 and Szigeti et al. 2013). In the QoS configuration proposed in this thesis, each ingress queue was assigned to a specific threshold value (See Appendix 1:Section 1. Part 3, 4 and 5, Section 2). Each value of the threshold holds a percentage of the ingress interface's buffer space. One of ingress queues was assigned to maximum queue threshold (the queue can hold up to its limit of frames at up to $\leq 100\%$ threshold). The other ingress queue was assigned to minimum queue threshold (the queue can hold up to minimum of queue frames at $< 100\%$ threshold). A high-threshold queue was configured as a high-priority queue (See Appendix 1:Section 2), where percentage of maximum queue threshold ≤ 100 (non-empty queue) and a low-threshold queue was configured as a low priority queue, where minimum queue threshold < maximum queue threshold.

A buffer reservation technique has been used to increase buffer size along with implemented parallel nodes of buffers to increase buffer speed performance. A switch buffer's memory space was divided between the switch common memory pool, the SVI, and the queue reserved pool (see Figure 5.10). a specific buffer memory space was defined for each queue, including ingress and egress thresholds. Packets were divided between two ingress and four egress queues via configured queue-sets. A buffer distribution scheme was implemented to reserve more space for each egress buffer.

Thus, all buffers cannot be consumed by one egress queue, and the system can manage whether funding buffer space to demand queue. The remaining free common pool interfaces were set to reserve up to 50% of the available switch memory pool.

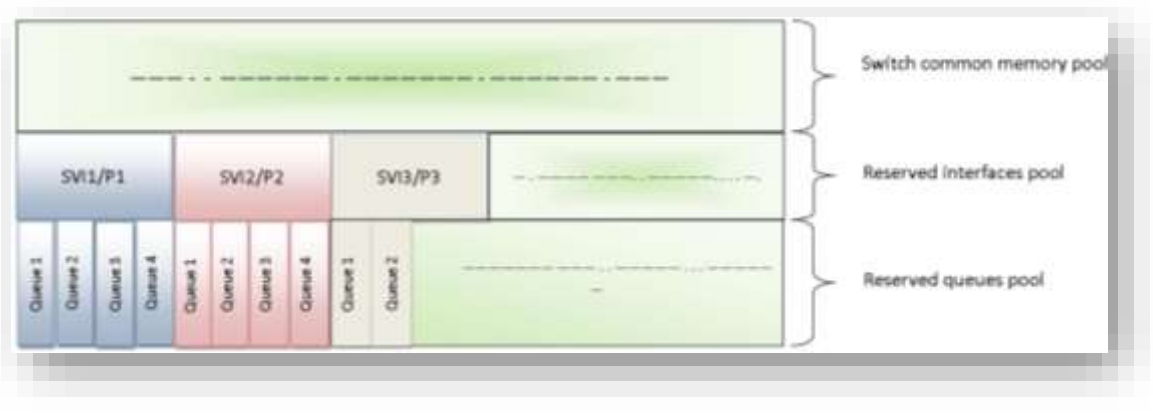


Figure 5. 10: : Novel egress queue buffer reservation.

A switch identifies whether the target queue has consumed less buffer space than its reserved volume (under-limit), whether the target queue has consumed all of its reserved amount (over-limit), and whether the switch memory pool is empty or not (no free buffers and free buffers, respectively). If the queue is not over-limit, the switch can reserve a buffer space from the interface pool or from the switch common memory pool. If no more space is available on the common pool or if queues are over-limit, the switch drops the packet.

After traffic has been classified, marked, and policed in two ingress queues, each packet is processed into four output queues that implement parallel NIDPS nodes. QoS also offers Shaped or Share Round Robin (SRR) technologies, which can vary the bandwidth, provided for the queues in the interface and control the rate at which packets are sent (Szigeti et al. 2013:260 and Cisco 2016a:840). The Shaped function SRR can guarantee each queue a bandwidth limit, but queues cannot share with each other if one or more queues reach their bandwidth limits. The Share function SRR can guarantee a bandwidth limit for each queue, and the other queues can share with each other if one or more queues reach their bandwidth limits. This research utilised Share in the ingress queues and Shaped in the first three egress queues. In the fourth queue, the Share mode was set to allow the queue to share process traffic with other available output queues (See Appendix1: Section 1. Part 4 and 5 and Section 2).

Queue technology is placed at specific points in Cisco switches to help prevent congestion (Szigeti et al. 2013 and Cisco 2016a). The total inbound bandwidth of all interfaces may exceed a ring

space of internal bandwidth. After packets are processed through classification, policing, and marking, and before packets pass into the switch fabric, the system allocates them to input queues. Because multiple input queue interfaces can simultaneously send packets to output queue interfaces, outbound queues are allocated after the internal ring in order to avoid congestion. The SRR ingress queue transmits packets to the internal ring, while the SRR egress queue sends the packets to the output link.

The novel configurable architecture has a large limit of buffer space and a generous bandwidth allocation for each queue. One of the ingress queues was set as a priority queue, which allowed the system to prioritise packets with particular DSCP values and thereby allocate a large buffer. It also allows buffer space to be used more frequently, and then adjusts the thresholds for each queue so that packets with inferior priorities are dropped when queues are full. This allows the system to ensure that high priority traffic is not dropped.

For more information, when the traffic comes to the interface, the packets are buffered in the priority ingress buffer (priority queue) and if the priority buffer is getting full, the traffic will transmit to the second ingress buffer. If all ingress buffers are getting full, the packets will be dropped or the switch can reserve another ingress buffer (which has the same priority) up to n, where n is the maximum number of ingress buffers. The formulae apply in one interface when there are two ingress queues and four egress queues.

When the λ_{in_i} is the arrival rate of packets, the rate for each ingress queue in one interface is $\lambda_{in_i}/_2$ .The output rate of arrival packets is $\lambda_{\mathrm{in_i}}/_4$ for each queue j in an interface i. The output rate (λ_{out}) of packets for one link is $\sum_{i=1}^{4} \lambda_{in_i}/_4$. The arrival traffic rate (λ_{in}) is $\sum_{i=1}^{n} \lambda_{in_i}$; The λ_{in} is the highest arrival traffic rate when *i=n*; and *n* is the maximum number of output links (interfaces). At the highest supported arrival rate λ_{in}, the maximum number of output queues will have been configured and the output rate for each output queue is $\lambda_{in}/_4/_n = \lambda_{in}/_{n4}$. The highest output traffic rate of n packets (λ_{out}) is $\sum_{\mathrm{i}=1}^{n}\left(\sum_{\mathrm{j}=1}^{4}\left(\lambda_{in}/_{n4}\right)\right)$.

In other hand, when the arrival packets pass through ingress buffers at speed rate (λ_{in_i}), the traffic (packets) speed (λ_{in_i}) was re-controlled as an interface speed limit (μ) (group of bytes per second) to organise and improve performance of ingress traffic speed, and then the traffic is stored in its range space, where packets were arranged and managed to exit the interface through egress queues (See Appendix 1 Section 1. Part 2).

Every arrival packet needs to be sent out of an ingress interface and then placed in egress buffers which permit an interface to hold packets when there are more packets to be transmitted than can physically be sent (experiencing congestion). If the switch cannot allocate enough buffers to hold all incoming traffic, the packets will be dropped. Availability of egress buffers determine if a packet is transmitted or not. When it comes to reducing packet drop, the switch does not concern itself with packets. Rather, it is concerned with the number of requested (reservation) buffers to which unbuffered packets can be added. The volume of egress buffering differs from platform to platform. Typically, two (2) reservation pools are available for Layer 3 network switches. These pools are the SVI reserved pool and the switch memory common pool. The switch memory common pool is used when the SVI reservation pool has previously been consumed.

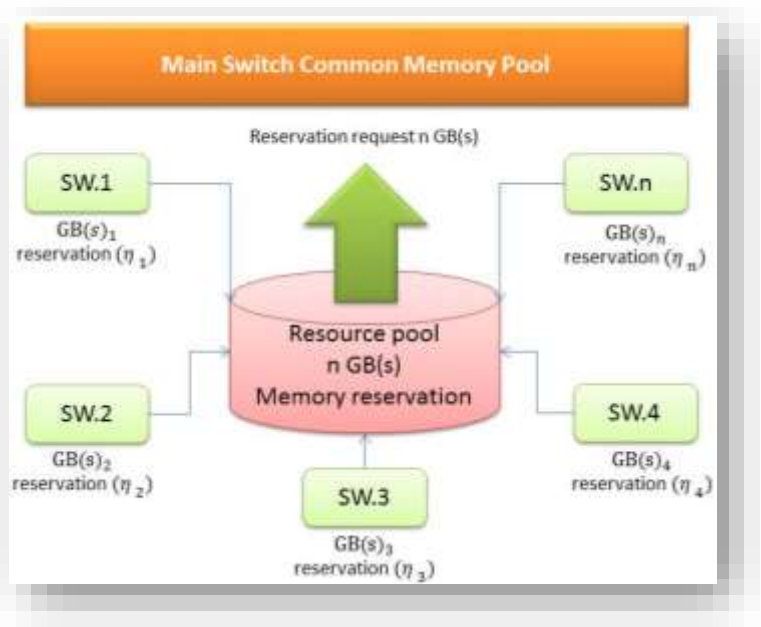


Figure 5. 11: Reservation buffer from n node of switches (η_n).

Furthermore, when packets (traffic) go through the output queues, the switch reserves buffers from the SVI reservation pool for all egress buffers and then if one egress buffer is fully consumed, the consumed egress buffer can reserve buffer space from the available buffers of other egress queues. When the SVI reservation pool is consumed by all egress buffers, it reserves more buffer from the switch memory command pool (see Figure 5.10). If all the switch's buffers are consumed, the packets will be dropped because no more space will be available.

All ingress and egress buffers above are collectively called one node of the switch's buffer (η_i). The common memory switch pool is the "holdup" storage area, where packets are held until they

can be processed. If the holdup area is full, more reservation buffer can be achieved by reserving memory from another switch memory pool in the network (see Figure 5.11) and even can be from outside networks. However, all egress buffers were controlled to limit rate β_k (kernel buffer speed) to prevent host based dropped packets. The kernel buffer rate (speed) should be the same as output queue.

In sensible implementations, limited buffers always have been implemented. If all the buffer space is already consumed, the arrival of additional packages causes buffer overflow. A buffer's queues, priority and threshold methods are part of the classification mechanisms that are responsible for dealing with the excess traffic. One of the possible measures taken because of buffer overflow is dropped packets.

5.4 Conclusion

NIDPSs are often unable to detect or prevent all unwanted traffic or malicious activities when traffic comes in at high-speeds and volumes. As a solution, this chapter has described a novel architecture that exploits Layer 3 Cisco switch technology, combined with parallel NIDPS nodes, to create queues with specific buffer and bandwidth sizes. The system thus increases queue buffer size automatically up to a network limit. It also services buffer space from an available queue buffer, port buffer, or switch pool memory buffer in order to hold more packets. This allows the system to organise and increase the processing speed of arriving packets by setting a number of parallel queues to be processed by parallel NIDPS nodes. The number of parallel processing NIDPS nodes needed in any particular system depends on network arrival rates. Therefore, it was necessary to operate with the class and quality of service technologies within the network switch. QoS DiffServ, including CoS and DSCP values, and a buffer reservation method were exploited in the proposed architecture which aims to reduce dropped buffer packets in egress queuing traffic in order to improve NIDPS performance. Property categories such as queue technology including ingress and egress, priority queue and thresholds, queue bandwidth, classification and policy methods including ACLs, buffer queue memory reservation and switch memory pool reservation are also important factors in the proposed solutionAn assumption is that there will be an underlying parallel implementation of the target destination (NIDPS in this case) and for each egress buffer commissioned there will be a port to a parallel node of the target system This will enable better performance and higher volumes of traffic to be processed successfully.The evaluation of the proposed solution is presented in the next chapter.

CHAPTER 6: EVALUATION OF THE PROPOSED SOLUTION

6.1 Introduction

This chapter presents an evaluation of the proposed solution of using a novel QoS architecture with parallel technology to improve NIDPS performance. The research uses Snort IDS in network based IDPS, which is configured to three modes (NIDPS analysis mode, detection mode and prevention mode). Experiments were conducted to test Snort NIDPS performance under high-speed traffic. It was demonstrated that Snort NIDPS performance (analysis, detection and prevention) rate can be increased by using the proposed solution

6.2 Evaluation of the Solution

In this section, two types of experiments were ran for each NIDPSs mode, one to test Snort's performance in terms of throughput without the support of the proposed solution (QoS and parallel technologies) and one with the proposed solution. Each experiment tested Snort NIDPS throughput when analysing traffic such as TCP/IP headers and when detecting or preventing unwanted traffic or malicious packets in a high-speed traffic.

6.2.1 Summary of experiments for evaluating the solution

The experiments evaluation carried out are shown in Table 6.1, along with their purpose.

Table 6. 1: Summary of experiments

Experiment Number	Purpose	Section in Thesis
Evaluate QoS technology proposed solution		
13	Evaluate the proposed solution to NIDPS analysis mode under high-speed traffic	6.2.2
14, 15, 16 and 17	Evaluate the proposed solution to NIDPS detection mode under high-speed ICMP, UDP and TCP traffic and malicious packets.	6.2.3
18	Evaluate different rules to detect malicious packets under high-speed traffic	6.2.4
19 and 20	Evaluate the proposed solution to NIDPS prevention mode under high-speed TCP/IP traffic and malicious packets	6.2.5
Evaluate QoS and parallel technologies proposed solution		
21	Evaluate the proposed solution of using QoS and parallel technologies together under high-speed traffic	6.3.1
22	Evaluate the system under 8Gbps traffic speed	6.3.2

6.2.2 Evaluate NIDPS analysis mode (NID-mode)

In this section, Snort NIDPS has been configured to sniffing mode. TCP/IP traffic been sent at speed 1ms (packets trip at 1 ms intervals). Two tests were done, the first one tested Snort without QoS and the second one test Snort with QoS.

6.2.2.1 Experiment 13: Snort with and without QoS to analyse TCP/IP header in high-speed traffic

In this experiment more than 38,000 packets (TCP/IP traffic) has been sent. Each packet was 1KB in size and the interval between each transmission was 1ms.

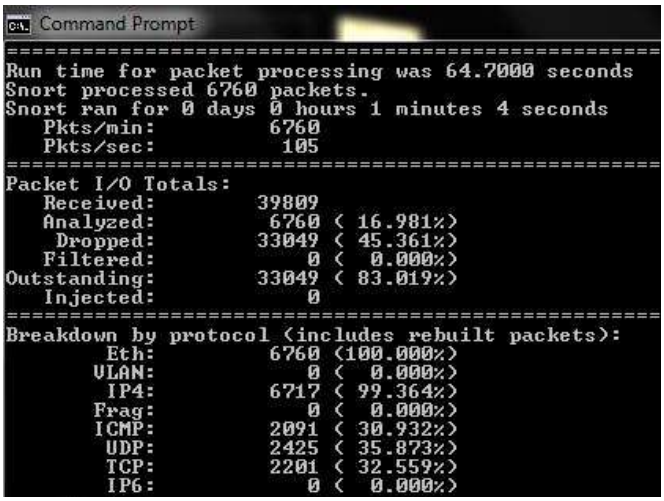


Figure 6. 1: Snort NIDPS without QoS in 1ms

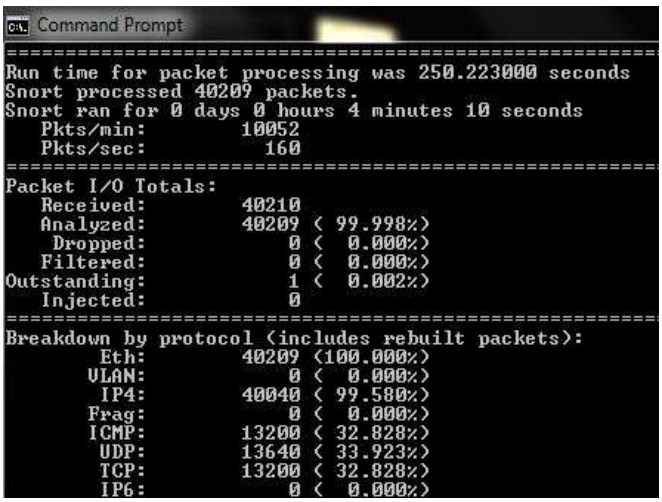


Figure 6. 2: Snort NIDPS with QoS in 1ms.

Table 6. 2 : Snort with QoS reaction to TCP/ IP Header in high-speed traffic.

Test type	The number of Packets received	Total packets analysed of total the packets received	Packets dropped of total packets received	Packets filtered of total packets received	Packets outstanding of total packets received	Packets rejected of total packets received	% packets analysed	% packets dropped	% packets outstanding
Without QoS	39809	6760	33049	0	33049	0	16.981%	45.361%	83.019%
	Snort Processor Times = 64s -> (Pkts/min:6760 - Pkts/sec:105)								
With QoS	40210	40209	0	0	1	0	99.998%	0.00%	0.002%
	Snort Processor Times = 250s-> (Pkts/min:10052 – Pkts/sec:160)								

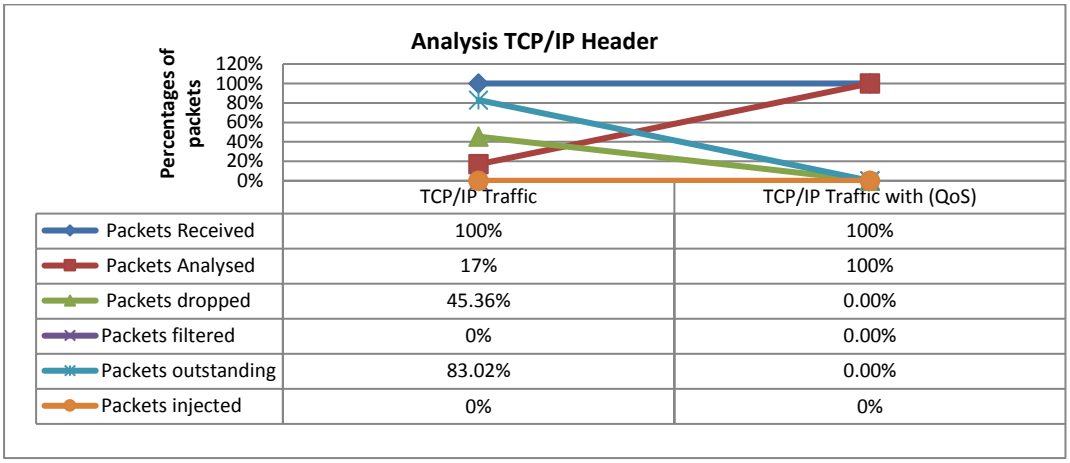


Figure 6. 3: Snort NIDPS with QoS reaction to TCP/IP in 1ms.

As the results in Figures 6.1, 6.2 and 6.3 show, all packets that were sent have been received by the machines. Figure 6.1 demonstrates that Snort without QoS analysed just less than 17% of the traffic that reached the wire with more than 45% dropped and more than 83% outstanding (see Table 6.2 and Figure 6.3). When the same experiment was run with QoS, Snort analysed nearly 100 % of the total packets that it received with 0 % dropped and 0.002% outstanding packets (see Figure 6.3). The results show that Snort performance analysis is significantly improved when using QoS technology.

6.2.3 Evaluate NIDPS detection mode (NID-mode)

In this section, we ran four experiments, each to test Snort's detection rate with and without support a QoS technology for a particular type of header or packet. The headers and packets were ICMP, UDP, TCP and malicious packets sent in speedy traffic.

6.2.3.1 Experiment 14: Snort with QoS reaction to detect ICMP header in high-speed traffic

In this experiment, more than 38,000 IP/ICMP packets were sent in interval traffic (packets) speed of 1ms. Each packet carried 1Kbyte. The aim was to detect IP/ICMP packets.

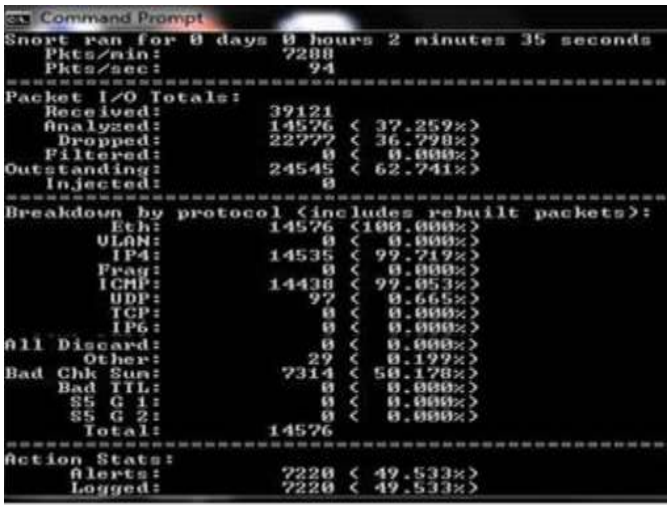


Figure 6. 4: Snort detects ICMP header in 1ms.

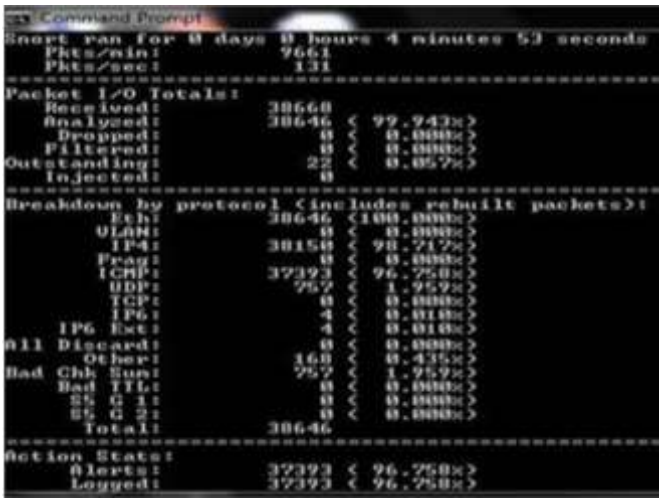


Figure 6. 5: Snort with QoS detects ICMP header in 1ms.

Table 6. 3: Snort with QoS reaction to ICMP Header in high-speed traffic.

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	ICMP packets alerts	ICMP packets logged	% packets alerts	% Packets logged
Without QoS	39121	37.259%	100%	14438	0	97	7220	7220	50.007%	50.007%
	Snort Processor Times = 155s -> (Pkts/min:7228 – Pkts/sec:94)									

With QoS	38668	99.943%	100%	37393	0	757	37393	37393	100.00%	100.00%
	Snort Processor Times = 293s -> (Pkts/min:9661 – Pkts/sec:131)									

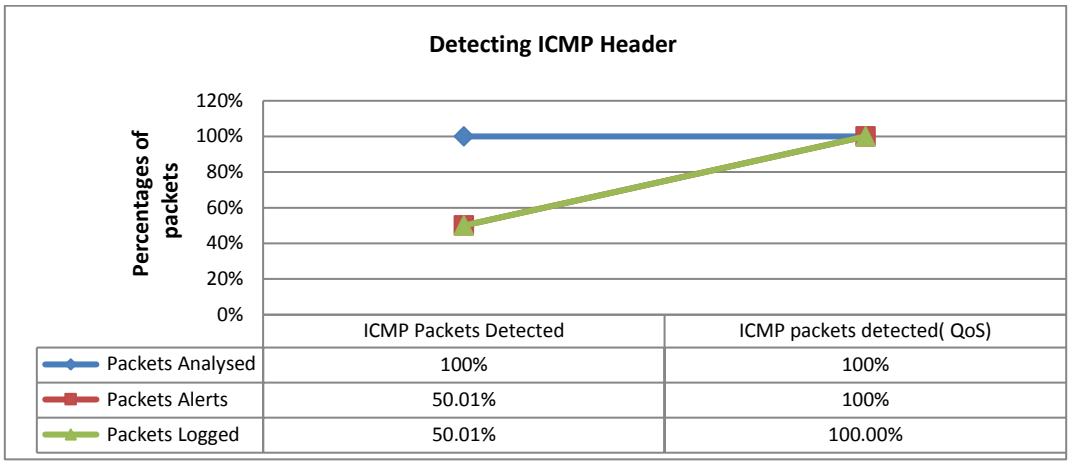


Figure 6. 6: Snort with QoS reaction to detect ICMP packets in 1ms.

As the results show in Figures 6.5, 6.5 and 6.6, when more than 38,000 IP/ICMP packets were sent with an interval time of 1ms, Snort alerted and logged 7220 of the 14,438 ICMP packets that were analysed (see Figure 6.6 and Table 6.3). When the same number of packets was sent at the same speed, using QoS, Snort detected all of the ICMP packets that it analysed (see Figure 6.5 and 6.6). This experiment shows that when Snort NIDPS was used without QoS, it only detected 50% of the total packets analysed, but when Snort was used with QoS, Snort detected 100% of the total packets that it analysed (see Figure 6.6).

6.2.3.2 Experiment 15: Snort with and without QoS to detect UDP headers in high-speed traffic

In this experiment, more than 38,000 IP/UDP packets were sent at interval traffic speed of 0.5ms. Each packet carried 1Kbyte. The aim was to detect UDP packets.

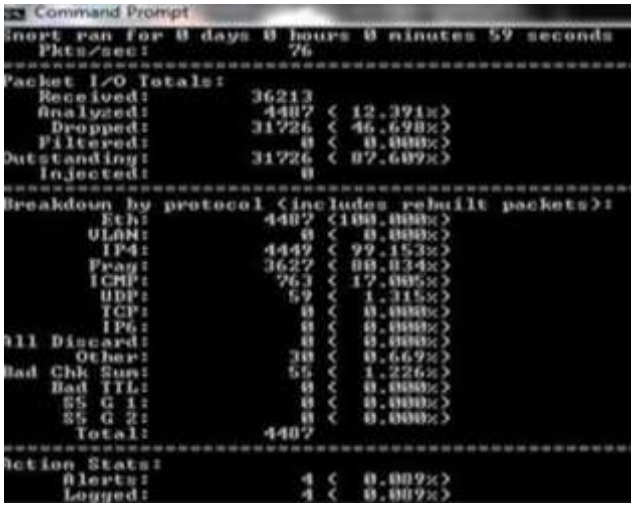


Figure 6. 7: Snort detects UDP header in 0.5ms.

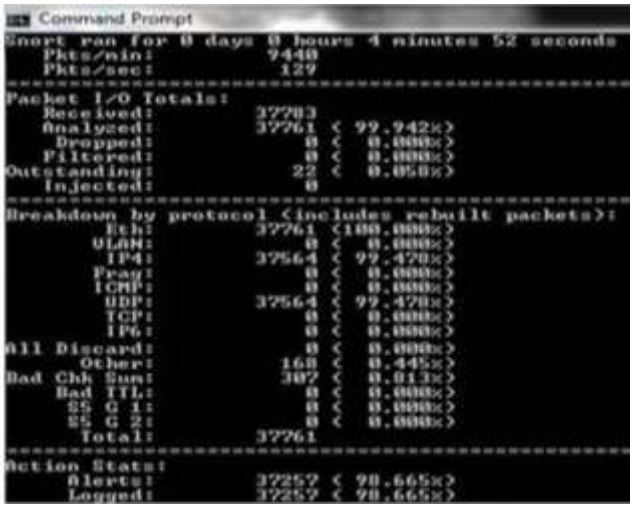


Figure 6. 8: Snort with QoS detect UDP header in 0.5ms.

Table 6. 4: Snort with QoS reaction to UDP header in high-speed traffic.

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	UDP packets Alerts	UDP packets logged	% packets alerts	% packets logged
Without QoS	36213	12.391%	100.00%	763	0	59	4	4	6.780%	6.780%
	Snort Processor Times = 59s -> (Pkts/sec:76)									
With QoS	37783	99.942%	100.00%	0	0	37564	37257	37257	99.182%	99.182%
	Snort Processor Times = 292 -> (Pkts/min:9440 – Pkts/sec:129)									

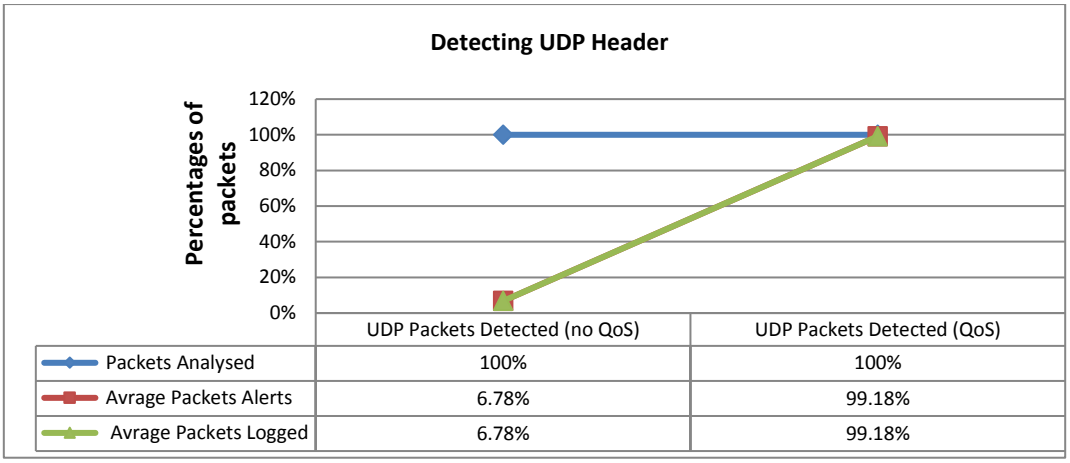


Figure 6. 9: Snort with QoS reaction to detect UDP packets in 0.5ms

As the results show in Figures 6.7, 6.8 and 6.9, when the traffic (packets) was sent in an interval time 0.5ms, Snort only alerted and logged nearly 4 of the 59 UDP packets that were analysed (see Table 6.3). It detected fewer than 7% of all UDP packets that it analysed (see Figure 6.7 and 6.9). When Snort was used with QoS and was sent the same traffic and speed at 0.5ms, Snort detected more than 99% of the total UDP packets analysed (see Figure 6.9 and Table 6.4). This experiment shows that the Snort NIDPS performance detection improved from 7% to 99% when the QoS configuration was used (see Figure 6.9).

6.2.3.3 Experiment 16: Snort with and without QoS to detect TCP header in high-speed traffic

In this experiment, more than 38,000 TCP/IP packets were sent at interval packets speed of 0.5ms. Each packet carried 1Kbyte. The aim was to detect TCP packets.

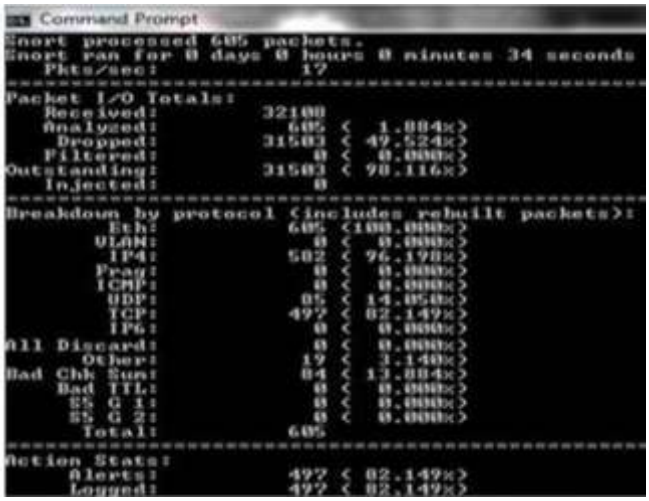


Figure 6. 10: Snort detects TCP headers in 0.5ms.

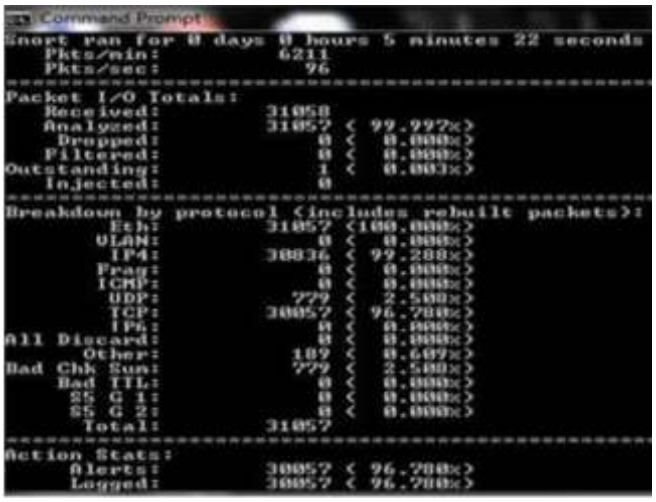


Figure 6. 11: Snort with QoS detect TCP header in 0.5ms.

Table 6. 5: Snort with QoS reaction to TCP Header in high-speed traffic.

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	TCP packets Alerts	TCP packets logged	% Packets alerts	% Packets logged
Without QoS	32108	1.884%	100.00%	0	497	85	497	497	100.00%	100.00%
	Snort Processor Times = 34s -> (Pkts/sec:17)									
With QoS	31058	99.997%	100.00%	0	30057	779	30057	30057	100.00%	100.00%
	Snort Processor Times = 322s -> (Pkts/min:6211 – Pkts/sec:96)									

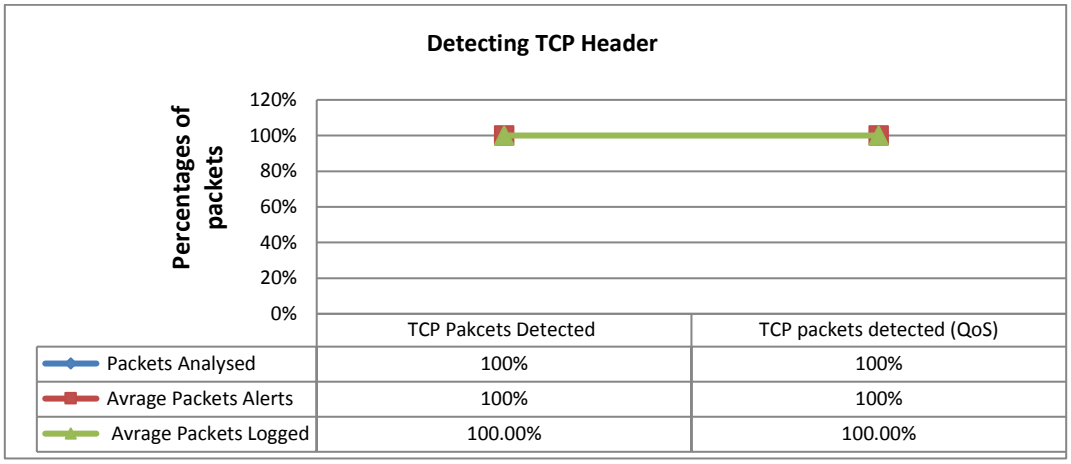


Figure 6. 12: Snort with QoS reaction to detect TCP packets in 0.5ms.

Figures 6.10, 6.11 and 6.12 show that Snort detected 100% of the total TCP packets that it is analysed, even without QoS (see Table 6.5 and Figure 6.12).

6.2.3.4 Experiment 17: Snort with and without QoS to detect malicious packets in high-speed traffic

In this experiment, flood traffic was generated with UDP malicious packets (threads) in traffic speed (60000Bps of flooded traffic with 225 threads sent at an interval time of 1 mSec). Two tests were conducted: one test of Snort without QoS and one of Snort with QoS.

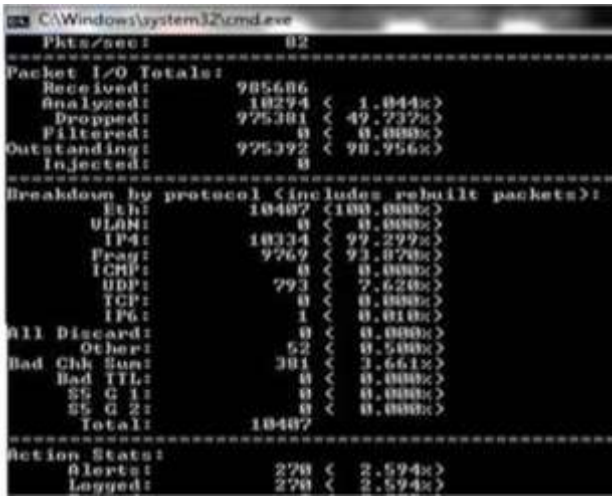


Figure 6. 13: Snort detects malicious packets in high-speed traffic.

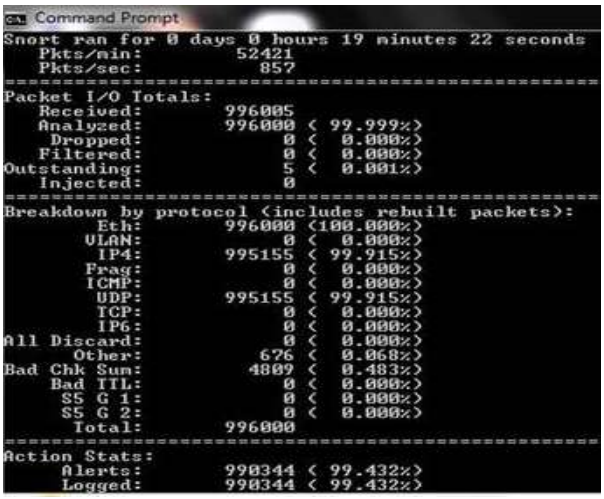


Figure 6. 14: Snort with QoS detects malicious packets in high-speed traffic.

Table 6. 6: Snort with QoS reaction to malicious packets in high-speed traffic.

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	UDP malicious packets Alerts	UDP malicious packets logs	% packets alerts	% packets logged
Without	985686	1.004%	100.00%	0	0	793	270	270	34.048%	34.048%

QoS	Snort Processor Times = 125s -> (Pkts/min:5090 – Pkts/sec:82)									
With QoS	996005	99.999%	100.00%	0	0	995155	990344	990344	99.517%	99.517%
	Snort Processor Times = 19.22m -> (Pkts/min:52421 – Pkts/sec:857)									

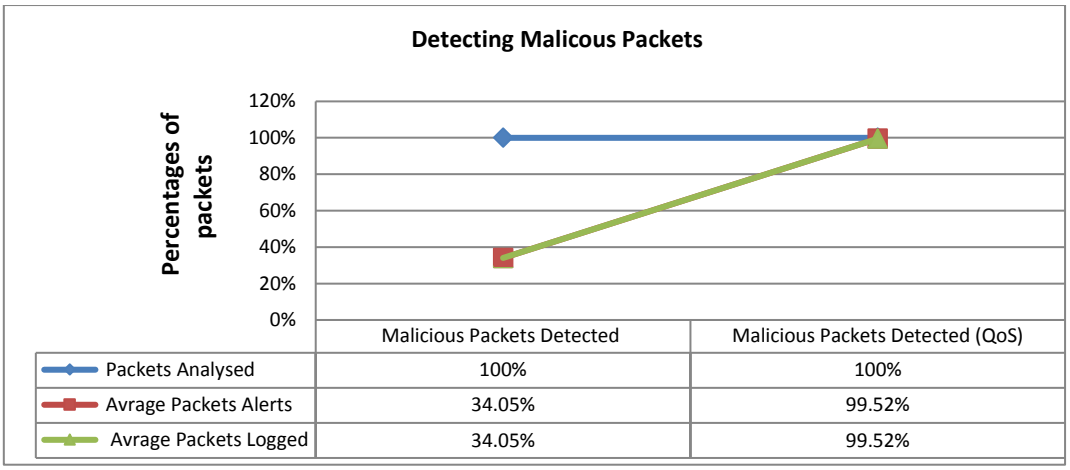


Figure 6. 15: Snort with QoS reaction to detect malicious packets in high-speed traffic.

6.2.3.5 Summary of Detection Experiments

As the results show in Figures 6.13, 6.14 and 6.15, when malicious traffic was sent at high-speeds and values, Snort detected 270 of the 793 malicious packets that it analysed. It detected fewer than 35% of the total malicious packets analysed (see Table 6.6). When the same traffic was generated with the same speed and value, but Snort was supported by QoS, Snort detected more than 99% of the total malicious packets that it analysed (see Figure 6.15 and Table 6.6). This experiment showed that the Snort NIDPS performance detection improved while QoS technology was used.

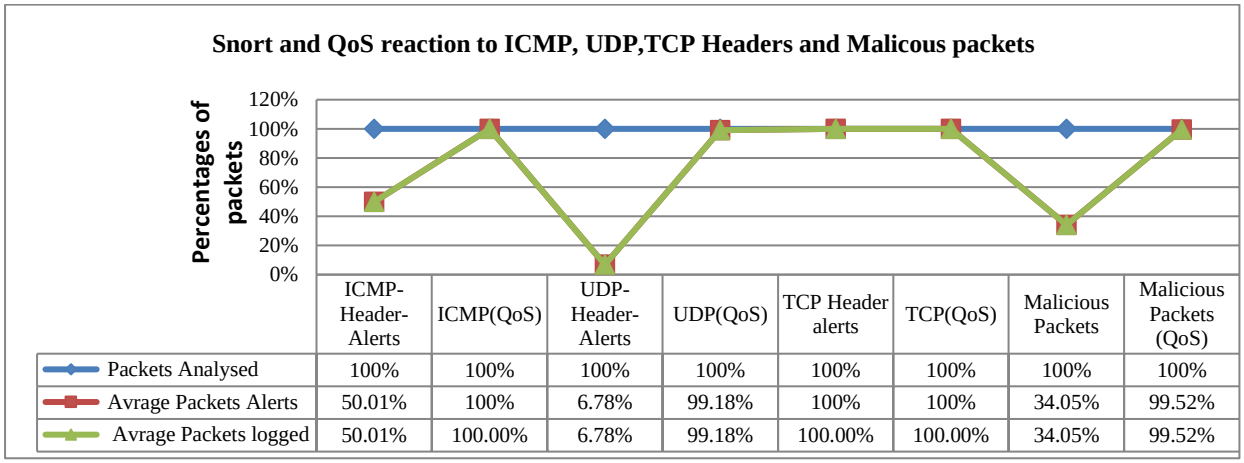


Figure 6. 16: Snort and QoS reaction to ICMP, UDP, TCP and malicious packets in high-speed traffic.

As a summary for experiments 14, 15, 16 and 17see Figure 6.16 which shows that Snort analysed every single packet that reached the wire. When IP traffic was sent at an average of 35000 packets per 1ms, Snort lost alerts and logs. For ICMP headers, Snort lost more than 49 % of the total ICMP packets that it is analysed and it missed more than 96 % of the total UDP packets analysed, but and for TCP , Snort alerted 100 % of the total TCP packets analysed (see Figure 6.16) . With the proposed new QoS architecture, Snort alerted 100 % of ICMP packets, more than 99 % of UDP packets and 100 % of TCP traffic (see Figure 6.16). Also Figure 6.11 shows that Snort lost more 66% of the total malicious packets that it is analysed, but when QoS is used, Snort detected more than 99% of the total packets that it analysed. The experiments show that Snort NIDPS detection performance improved when supported by QoS configuration.

6.2.4 Evaluation of different Snort NIDPS rules

In this section experiments are presented which evaluated Snort's capability with different types of rules to detect malicious packets with and without QoS. In these experiments, malicious packets (threads) have been generated in high-speed traffic by using NetScanPro, WinPcap, Packets Flooder and Packets Traceroute tools. Every single test was repeated 3 times with and without QoS to get the average number of packet alerts and logs. Different Snort rules have been tested under high-speed traffic. Flood traffic has been sent at 65000Bps with different types of 255 threats at interval time 1mSec.

6.2.4.1 Experiment 18: Snort rules with QoS reaction to detect malicious packets in high-speed traffic.

In experiment 18, six action rules (ttl, content, hexadecimal content, offset depth and dsize) have been tested to show Snort NIDPS performance detection with and without QoS.

Test 1: Time To Live (TTL) rules

The TTL rule detects any packet matched to the specified ttl value. The ttl keyword takes numbers from 0 to 255. In this experiment, 255 UDP malicious packets have been sent with ttl 128 at 1mSec with flooding traffic at 65000Bps. The following rule was written to check if ttl of the UDP packets is equal to 128. If a packet matches the rule, Snort will provide an alert packet.

Alert udp any any -> any any (msg:" Check Time To Live value in udp header"; ttl: 128; sid: 10000171 ;).

Table 6. 7: Snort ttl keyword rule reaction to malicious packets in high-speed traffic without QoS

The number of Packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packet alerts	UDP malicious packets logged	% packets alerts	% packets logged
985686	1.044%	100.00%	0	0	793	270	270	34.047 %	34.047 %
987885	1.075%	100.00%	0	0	796	265	265	33.291 %	33.291 %
993061	1.088%	100.00%	0	0	732	239	239	32.650 %	32.650 %
The Average of Snort Processor Times : (125s+126s+126s) / 3 = 125.6s -> (Pkts/min:5308 – Pkts/sec:84)									
The average of UDP Malicious Packets logged = 33.37%									
The average of UDP Malicious Packets alerts = 33.37%									

Table 6. 8: Snort ttl keyword rule with QoS reaction to malicious packets in high-speed traffic Without QoS

The number of packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packets alerts	UDP malicious packets logged	% packets alerts	% packets logged
964952	99.995%	100.00%	47	0	964012	961539	961539	99.743 %	99.743 %
996005	99.915%	100.00%	0	0	995155	990344	990344	99.516 %	99.516 %
918372	99.999%	100.00%	0	0	917589	913523	913523	99.556 %	99.556 %
The Average of Snort Processor Times : (18.52m+19.22m+17.54m) / 3 = 18.42m -> (Pkts/min:53349 – Pkts/sec:855)									
The average of UDP Malicious Packets logged = 99.605%									
The average of UDP Malicious Packets alerts = 99.605%									

Test 2: Content Keyword rule

Snort has a capability to catch a data pattern inside packets and headers. The pattern can be expressed as an ASCII string from (American Standard Code for Information Interchange) or as a hexadecimal number. For example, some malicious attacks have signatures and the content rule can detect them. The following rule was created to detect a pattern “abcdef” in the data part of all UDP packets.

Alert udp any any -> any any (msg:” Check or to find data pattern inside packets”; ttl: 128; content: “abcdef”; Sid: 100000172 ;).

In this experiment, 255 UDP malicious packets were sent at 1ms intervals with flood traffic 65000Bps. The malicious packets were sent are including "abcdef" with ttl 128.

Table 6. 9: Snort content keyword rule reaction to malicious packets in high-speed traffic without QoS

The number of Packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packets alerts	UDP malicious packets logged	% Packets alerts	% Packets logged
988183	1.052%	100.00%	0	0	576	239	239	41.493 %	41.493 %
988109	1.069%	100.00%	0	0	564	232	232	41.134 %	41.134 %
987882	1.068%	100.00%	0	0	751	236	236	31.424 %	31.424 %
The Average of Snort Processor Times : (125s+125s+124s) / 3 = 124.6s -> (Pkts/min:5195 – Pkts/sec:83)									
The average of UDP Malicious Packets logged = 40.15%									
The average of UDP Malicious Packets alerts = 40.15%									

Table 6. 10: Snort content keyword rule with QoS reaction to malicious packets in high-speed traffic.

The number of packets received	The total Packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packets Alerts	UDP malicious packets logged	% Packets alerts	% Packets logged
999158	92.242%	100%	0	0	920850	919567	919567	99.860%	99.860%
999973	92.070%	100%	0	0	919855	918164	918164	99.816%	99.816%
1017310	99.999%	100%	0	0	1016423	1012125	1012125	99.577%	99.577%
The Average of Snort Processor Times : (19.33m+19.36m+19.51m) / 3 = 19.4m -> (Pkts/min:50168 – Pkts/sec:807)									
The average of UDP Malicious Packets logged = 99.751%									
The average of UDP Malicious Packets alerts = 99.751%									

Test 3: Content-hexadecimal Keyword rules

In this test encrypted UDP malicious packets were sent at speed 1mSec with flood traffic 65000Bps.The following rule was designed to the same Content rule, but the pattern is listed in the hexadecimal. The hexadecimal number 61 is equal to ASCII character (a), 62 is equal to (b), 63 is equal to (c) and 64 is equal to (d).

Alert udp any any -> any any (msg:" Check hexadecimal characters in side data"; ttl: 128; content: "|61 62 63 64|"; Sid: 100000173 ;).

Table 6. 11: Snort content-hexadecimal keyword rule reaction to malicious packets in high-speed traffic without QoS.

The number of Packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packets Alerts	UDP malicious packets logged	% packets alerts	% packets logged
990065	1.066%	100.00%	0	0	570	241	241	42.280 %	42.280 %
990071	1.049%	100.00%	0	0	572	237	237	41.433 %	41.433 %
986888	1.067%	100.00%	0	0	555	238	238	42.882 %	42.882 %
The Average of Snort Processor Times : (126s+124s+124s) / 3 = 124.3s -> (Pkts/min:5278 – Pkts/sec:83)									
The average of UDP Malicious Packets logged = 42.12%									
The average of UDP Malicious Packets alerts = 42.12%									

Table 6. 12: Snort content-hexadecimal keyword rule with QoS reaction to malicious packets in high-speed traffic.

The number of packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packets Alerts	UDP malicious packets logged	% packets alerts	% packets logged
1003040	99.999%	100%	0	0	1002190	997235	997235	99.505%	99.505%
1003428	99.999%	100%	0	0	1002583	997600	997600	99.502%	99.502%
1003200	99.999%	100%	0	0	1002355	997403	997403	99.505%	99.505%
The Average of Snort Processor Times : (19.31m+19.30m+19.31m) / 3 = 19.3m -> (Pkts/min:52801– Pkts/sec:856)									
The average of UDP Malicious Packets logged = 99.504%									
The average of UDP Malicious Packets alerts = 99.504%									

Test 4: Offset Keyword rules

The offset keyword rule can be used together with the content rule. It is used to examine a target signatures at a specific domain offset from the start of the data part of the packets. The following rule was made to start detecting for the characters "abcdef" after 100 bytes from the start of the payload of UDP packets.

Alert udp any any -> any any (msg:" Start research for the word "abcdef" after 100 bytes from the start of data"; ttl: 128; content: "abcdef"; offset: 100; Sid: 100000174 ;).

Table 6. 13: Snort offset keyword rule reaction to malicious packets in high-speed traffic without QoS

The number of packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packet alerts	UDP malicious packets logged	% packets alerts	% packets logged
990286	1.057%	100%	0	0	553	232	232	41.952%	41.952%
990631	1.072%	100%	0	0	544	235	235	43.198%	43.198%
985661	1.062%	100%	0	0	562	238	238	42.348%	42.348%
The Average of Snort Processor Times : (125s+125s+124s) / 3 = 124.6s -> (Pkts/min:5277 – Pkts/sec:84)									
The average of UDP Malicious Packets logged = 42.26%									
The average of UDP Malicious Packets alerts = 42.26%									

Table 6. 14: Snort offset keyword rule with QoS reaction to malicious packets in high-speed traffic.

The number of packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packet alerts	UDP malicious packets logged	% Packets alerts	% Packets logged
1000607	99.999%	100%	0	0	999753	998507	998507	99.875%	99.875%
1000516	99.999%	100%	0	0	999664	998348	998348	99.868%	99.868%
1000830	99.999%	100%	0	0	999976	998729	998729	99.875%	99.875%
The Average of Snort Processor Times : (19.31m+19.31m+19.31m) / 3 = 19.31m (Pkts/min: 52665 – Pkts/sec:854)									
The average of UDP Malicious Packets logged = 99.872%									
The average of UDP Malicious Packets alerts = 99.872%									

Test 5: Depth Keyword rules

To specify more check of matching limit in the pattern of data, the depth rule can be used together with the content rule. This depth rule is used to specify offset from data part starting. The data after that offset rule will not be checked for pattern matching. The depth rule defines the point after which Snort should stop examining the pattern inside the data. If offset and depth keywords are used together with the content keyword, a specific pattern matching within the range of data can be done. The following rule was written to find out the characters "abcdef" between characters 4 and 100 of the data part of the UDP packets

Alert udp any any -> any any (msg:" Start research for the word "abcdef" between characters 4 and 100 bytes of the data"; ttl: 128; content: "abcdef"; offset: 4; depth: 100; Sid: 100000175 ;).

Table 6. 15: Snort depth keyword rule reaction to malicious packets in high-speed traffic without QoS

The number of packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packet alerts	UDP malicious packets logged	% packets alerts	% packets logged
988438	1.059%	100%	0	0	564	236	236	41.843%	41.843%
989543	1.066%	100%	0	0	557	235	235	42.190%	42.190%
989471	1.067%	100%	0	0	571	234	234	40.980%	40.980%
The Average of Snort Processor Times : (125s+125s+125s) / 3 = 125s -> (Pkts/min:5233 – Pkts/sec:83)									
The average of UDP Malicious Packets logged = 41.66%									
The average of UDP Malicious Packets alerts = 41.66%									

Table 6. 16: Snort depth keyword rule with QoS reaction to malicious packets in high-speed traffic.

The number of packets received	The total packets analysed of the packets received	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packet alerts	UDP malicious packets logged	% packets alerts	% packets logged
999025	99.999%	100%	0	0	998134	994787	994787	99.664%	99.664%
1001853	100.000%	100%	0	0	1000963	996965	996965	99.600%	99.600%
1000624	99.999%	100%	0	0	999733	995775	995775	99.604%	99.604%
The Average of Snort Processor Times : (19.32m+19.31m+19.31m) / 3 = 19.31m - > (Pkts/min:52657– Pkts/sec:853)									
The average of UDP Malicious Packets logged = 99.622%									
The average of UDP Malicious Packets alerts = 99.622%									

Test 6: Dsize Keyword rules

Various malicious attacks are distributed a large size packets to the target system to cause buffer overflow which can be prevented by using the dsize rule. This rule is used to check if packets contain data of a length greater than, less than, or equal to a specified number. The following rule detects any UDP traffic, if the UDP packet size is over than 30000 bytes.

Alert udp any any -> any any (msg:" Check data size for packets"; ttl: 128; dsize :> 30000; Sid: 10000176 ;). In this experiment, UDP traffic has been sent at 65000Bps.

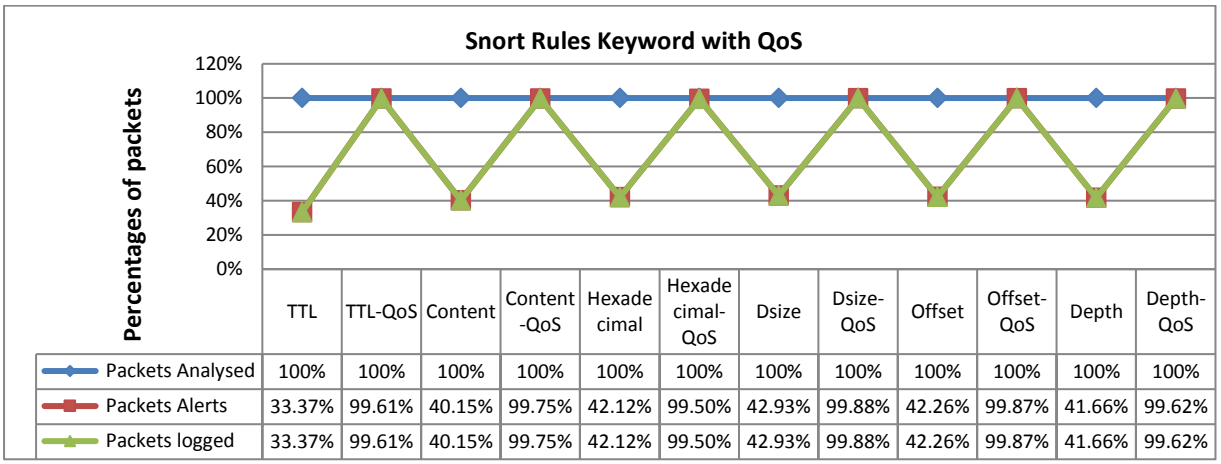
Table 6. 17: Snort dsize keyword rule reaction to malicious packets in high-speed traffic without QoS

The	The total	Total Eth	ICMP	TCP	UDP	UDP	UDP	%	%

number of packets received	packets analysed of the packets received	received of the total packets analysed	packets analysed	packets analysed	malicious packets analysed	malicious packet alerts	malicious packets logged	Packets alerts	Packets logged
989268	1.031%	100%	0	0	552	236	236	42.753%	42.753%
987379	1.058%	100%	0	0	559	244	244	43.649%	43.649%
989191	1.059%	100%	0	0	568	242	242	42.605%	42.605%
The Average of Snort Processor Times : (125s+125s+124s) / 3 = 124.6s -> (Pkts/min:5223 – Pkts/sec:83)									
The average of UDP Malicious Packets logged = 42.93%									
The average of UDP Malicious Packets alerts = 42.93%									

Table 6. 18: Snort dsize keyword rule with QoS reaction to malicious packets in high-speed traffic.

The number of packets received	The packets analysed of the total packets received	Total Eth packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP malicious packets analysed	UDP malicious packet alerts	UDP malicious packets logged	% packets alerts	% packets logged
999185	99.999%	100%	0	0	998332	997108	997108	99.877%	99.877%
1004432	99.999%	100%	0	0	1003579	1002351	1002351	99.877%	99.877%
1000284	99.999%	100%	0	0	999431	998209	998209	99.877%	99.877%
The Average of Snort Processor Times : (19.31m+19.34m+19.31m) / 3 = 19.32 -> (Pkts/min: 52699 – Pkts/sec:854)									
The average of UDP Malicious Packets logged = 99.877%									
The average of UDP Malicious Packets alerts = 99.877%									



	TTL	TTL-QoS	Content	Content -QoS	Hexade cimal	Hexade cimal-QoS	Dsize	Dsize-QoS	Offset	Offset-QoS	Depth	Depth-QoS
Packets Analysed	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
Packets Alerts	33.37%	99.61%	40.15%	99.75%	42.12%	99.50%	42.93%	99.88%	42.26%	99.87%	41.66%	99.62%
Packets logged	33.37%	99.61%	40.15%	99.75%	42.12%	99.50%	42.93%	99.88%	42.26%	99.87%	41.66%	99.62%

Figure 6. 17: Snort keyword rules and QoS reaction to detect malicious packets (threads) in high-speed traffic.

A summary of the rule tests of experiment 18 are shown in Figure 6.17, Snort analysed every single packet that reached the wire. When flood traffic was sent at an average speed of 65000 bytes with 255 UDP malicious packets per 1mSec, Snort loses alerts and logs for all rules that are used. Snort missed more than 66 % alerts of the malicious packets that were analysed for ttl rule; more than

59 % for content and depth rules; and more than 57 % for Content-hexadecimal, offset and dsize rules (see Figure 6.17). When QoS is used, Snort alerted more than 99 % of the malicious packet that were sent for all the rules (see Figure 6.17). Our experiments show that Snort detection performance has been improved when QoS technology is used.

6.2.5 Evaluate NIDPS prevention mode (NIP-mode)

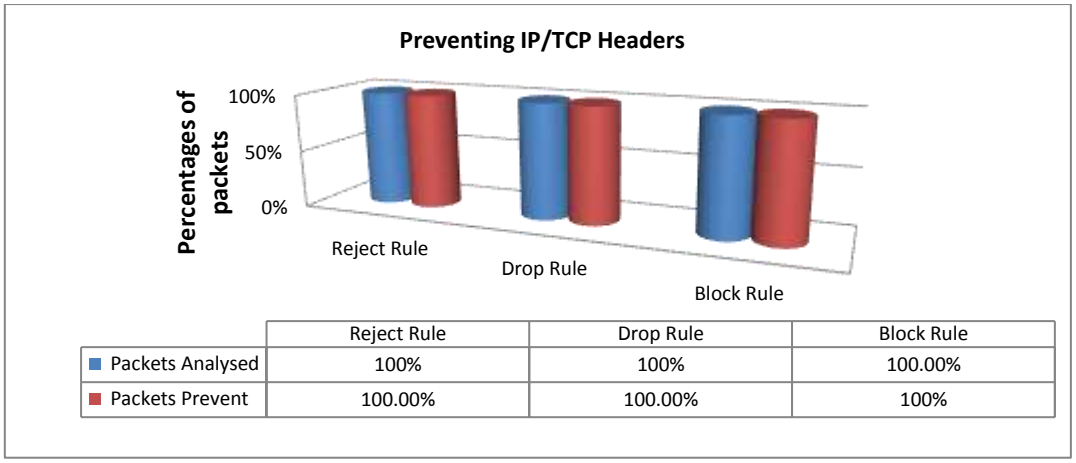
In this section, two experiments are presented which tested Snort NIDPS performance in prevention mode in high-speed traffic. In Experiment 19, Snort NIDPS was set to prevent TCP/IP traffic and in Experiment 20, it was set to prevent malicious packets (threads)

6.2.5.1 Experiment 19: Snort prevention rules with QoS reaction to prevent TCP/IP Header in High-Speed Traffic

In this experiment, Snort actions such as Reject, Drop and Block were tested in the task to prevent TCP/IP packets in high-speed traffic. More than 150000 TCP/IP packets were sent at interval speed of 1ms and each packet carried 1Kbyte. This experiment was also carried out in stage 1 of the experimental design but here it is carried out with QoS. However, the reason to return to this experiment (earlier presented in chapter 4, experiment 10, and section 4.5.2.) was to ascertain whether the output result of the Snort prevention mode test has the same results as the stage 1 experiment when QoS was used.

Table 6.19: Snort with QoS reaction to prevent TCP/IP Header in high-speed traffic.

Test type	Total packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	IP4 packets received	TCP/IP packets prevent	% packets prevent
Reject action rule	159649	100.00%	50000	53737	50095	153832	153832	100.00%
	Snort Processor Times = 28s -> (PKts/sec:5701)							
Drop action rule	150153	100.00%	50000	50000	50118	150118	150118	100.00%
	Snort Processor Times = 34s -> (PKts/sec: 4416)							
Block action rule	157745	100.00%	57625	50000	50092	157717	157717	100.00%
	Snort Processor Times = 29s -> (PKts/sec:5439)							



	Reject Rule	Drop Rule	Block Rule
Packets Analysed	100%	100%	100.00%
Packets Prevent	100.00%	100.00%	100%

Figure 6. 18: Snort reaction to prevent TCP/IP traffic in 1ms.

Our experiment shows that Snort NIDPS with QoS prevents all unwanted traffic in all NIDPS prevention action rules (see Figures 6.18 and Table 6.19). The percentages of prevent performance was the same as in experiment 10 but the speed of packets processing was increased to nearly 5701kpts/sec.

6.2.5.2 Experiment 20: Snort with QoS Reaction to prevent malicious packets in High-Speed Traffic

In this experiment, Snort prevention mode has been tested under high-speed traffic. Two tests were run, one Snort without QoS and one with QoS. IP traffic has been sent at 65000Bps with 255 malicious UDP threads in interval packets delay of 1 mSec.

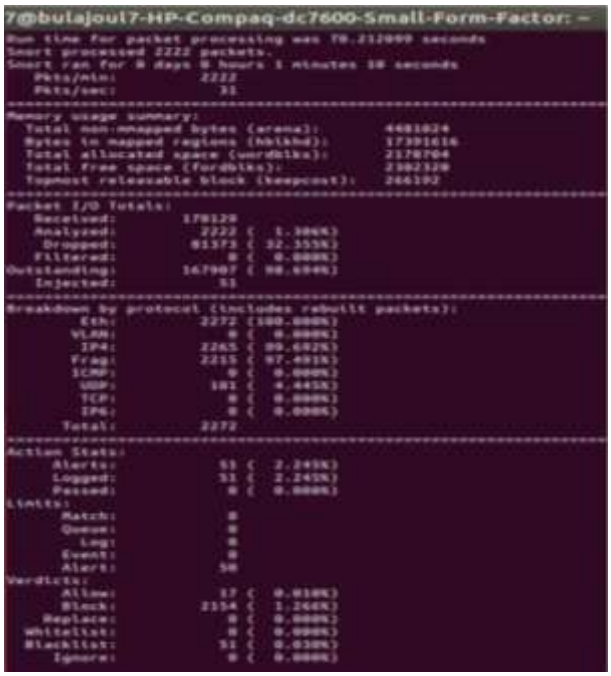


Figure 6. 19: Snort without QoS preventing malicious UDP packets in high-speed and heavy traffic.

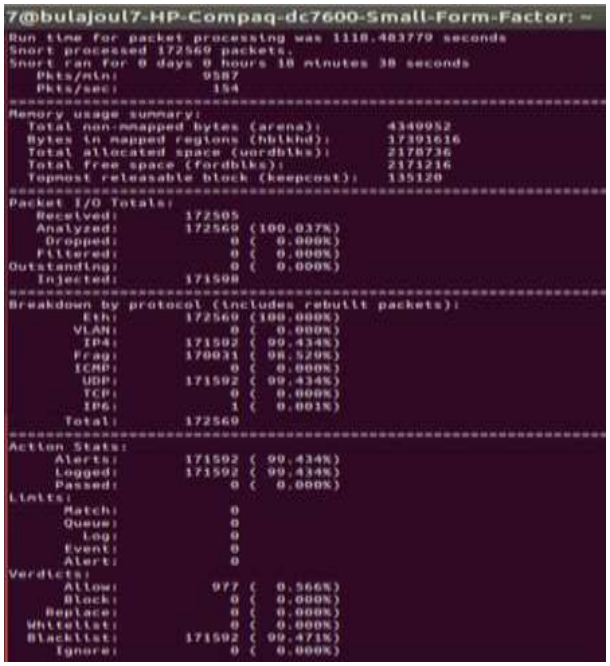


Figure 6. 20: Snort with QoS preventing malicious UDP packets in high-speed and heavy traffic.

Table 6. 20: Snort with and without QoS preventing malicious packets in high-speed traffic.

Test type	The Number of Packets received	Total Packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	IP4 packets received	% Malicious packets prevent	% Malicious packets prevent
Without QoS	170129	2222	100.00%	0	0	101	2265	51	50.495%
	Snort Processor Times = 70s -> (Pkts/min:2222 – Pkts/sec:31)								
With QoS	172505	172569	100.00%	0	0	1715925	141592	171592	100.00%
	Snort Processor Times = 18.28m -> (Pkts/min:9587 – Pkts/sec:154)								

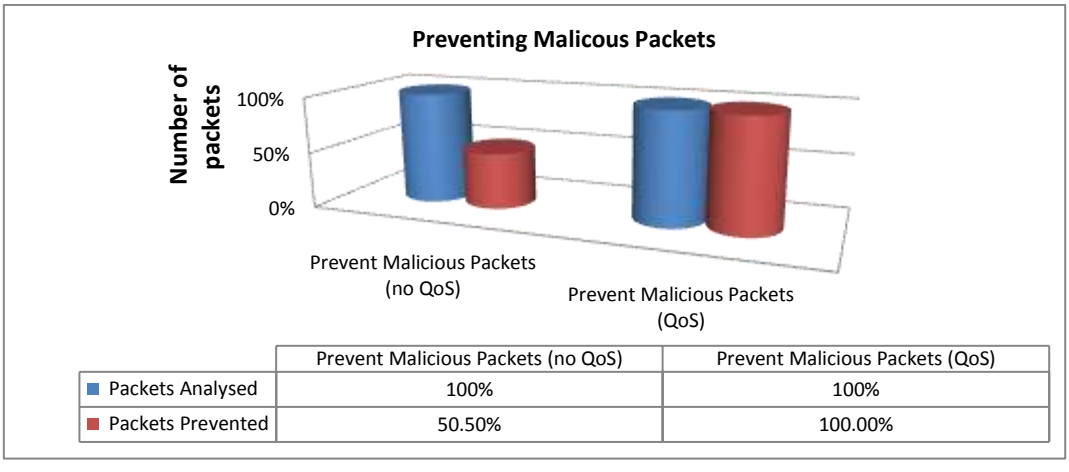


Figure 6. 21: Snort with QoS reaction to prevent malicious packets in high-speed traffic.

As the results show in Figures 6.19, 6.20 and 6.21, when malicious traffic was sent at high-speed and volume, Snort prevents 51 of the 101 malicious UDP packets that it analysed (see Table 6.20). It blocked fewer than 51% of the total malicious packets analysed (Figures 6.19 and 6.21). When the same traffic was generated with the same speed and value, but Snort was supported by QoS, Snort prevented 100% of the total malicious UDP packets that it analysed (see Figures 6.20 and 6.21 and Table 6.20). This experiment showed that the Snort NIDPS performance prevention rate is improved when QoS technology is used.

6.3 Parallel NIDPS with QoS technologies

All the experiments presented in this chapter so far tested NIDPS performance analysis, detection and prevention with and without QoS technology under high-speed traffic. The experimental results show that Snort performance is significantly improved when QoS configuration technology is used; and the experimental results show that Snort NIDPS packets processing performance (throughput) is improved as well (see Figures 6.1 to 6.21 and Tables 6.1 to 6.20). For example in experiment 20, Snort NIDPS was tested with and without support from QoS. Without QoS, Snort analysed less than 2% of the total packets received, detected less than 51% of the total packets analysed, and prevented less than 51% of the total packets analysed with packets processing speed at 2222 Pktspm and 31Pktsps (see Figures 6.19 and 6.22 and Table 6.20). When the QoS is used , Snort analysed 100% of total packets received, detected 100% of the total packets analysed, and prevented all unwanted traffic with improved packets processing speed of 9587Pktspm and 154Pktsps (see Figures 6.20 and 6.22 and Table 6.20). Also experiment 20 shows that when 172505 packets were sent, Snort just processes 2222 packets at running time 70s and with QoS, Snort processes most of packets that it received at running time 18.28m, giving a higher Pkt throughput rate (see Figure 6.22).

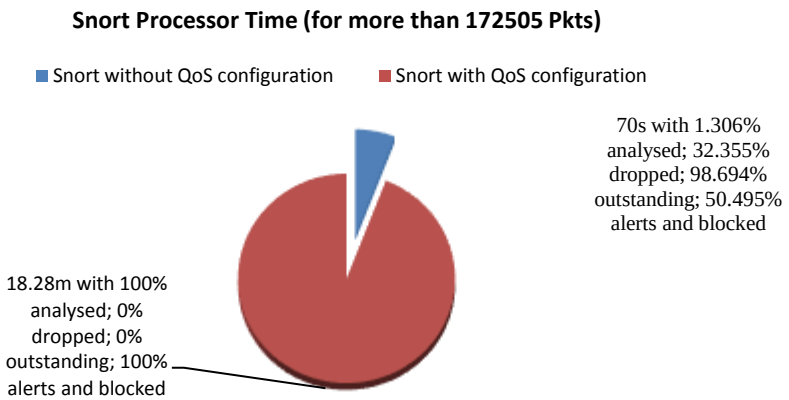


Figure 6. 22: NIDPS Processor Time (for 65000B sending per (s) with UDP 255 threads 1mSec).

As a solution to reduce Snort's overall processing running time and increase packets processing throughput further, it is proposed to use parallel NIDPS technology with QoS. Snort NIDPS has been configured from a single node system to a multi-node system. We configured and treated traffic using QoS management, which produced four output queues (see Figure 6.23). Then each queue was scanned individually using an access control list function (ACL). Traffic was filtered according to classification and different packages were sent to specific parallel Snort nodes.

6.3.1 Experiment 21: Parallel Snort NIDPS with QoS technologies

In experiment 13 (see section 6.2.2.1), a single node of Snort NIDPS was tested without any QoS treatment. Nearly 38,000 TCP/IP packets were sent at 1ms, each packet carrying 1kbyte. Snort analysed less than 17% of the total packets received in 64s with a processing rate of 105 Pktsps (see Table 6.2 and Figure 6.1), but when a single Snort NIDPS was run with a QoS configuration and sent the same packets Snort NIDPS analysed all the packets that reached the wire in 250s without dropping any and only having 1 packet outstanding yielding a processing rate of 160 Pkts/s (see Table 6.2 and Figure 6.2). In this experiment 21, the same number of packets was sent at the same speed and of the same size. Parallel NIDPS technology was used (in three queues). After configuring the switch using QoS, Snort analysed 100% of the packets in less time (103s) (see Figure 6.23 and Table 6.21) and increased packet processing rate to 389 Pkts/s (see Table 6.21 and Figure 6.24).

Table 6. 21: Parallel Snort with QoS.

Test type	The number of Packets received	Total packets analysed of total the packets received	Packets dropped of total packets received	Packets filtered of total packets received	Packets outstanding of total packets received	Packets injected of total packets received	% packets analysed	% packets dropped	% packets outstanding
Without QoS	39809	6760	33049	0	33049	0	16.981%	45.361%	00.002%
	Snort Processor Times = 64 -> (PKts/min:6760 - PKts/sec:105)								
With QoS	40210	40209	0	0	1	0	99.998%	0.00%	0.00%
	Snort Processor Times = 250s-> (PKts/min:10052 – PKts/sec:160)								
Parallel NIDPs With QoS	40005	40005	0	0	0	0	100.00%	0.00%	0.00%
	Snort Processor Times = 103 -> (PKts/min:40005 – PKts/sec:389)								

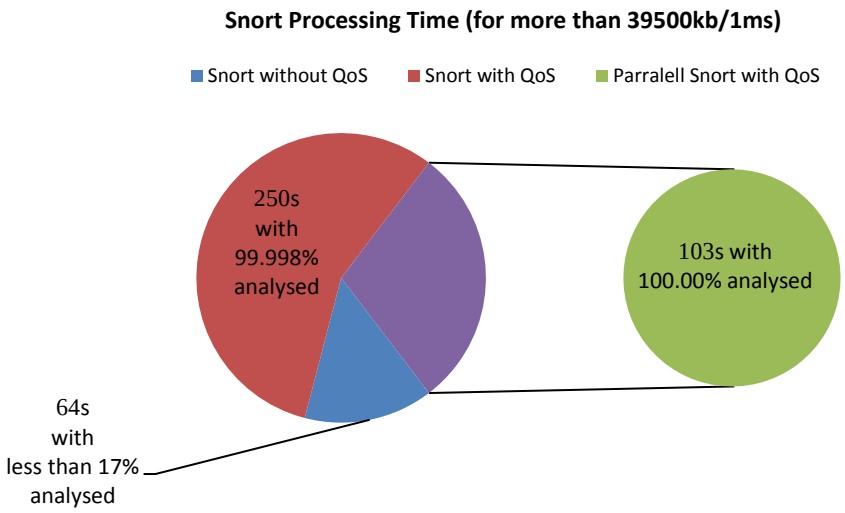


Figure 6. 23: Parallel Snort NIDPS Processing Time for 40,000kb sending at 1ms intervals.

The experiment also shows that, Snort's processing running time decreased from 250s to 103s (see Figure 6.24); and Snort's performance in packets processing throughput has been improved form 105 Pkts/s to 389 Pktsps, showing an improvement of around 60% or roughly 3 times speed up (see Figure 6.24). The experiments prove that Snort performance improves significantly using QoS and parallel NIDPs technology. It has processed more than 40,000KB in 103s with 0 percent dropped or outstanding (see Figures 6.23). Obviously speed up depends on the number of nodes and processors used and far greater speed up is possible with more nodes.

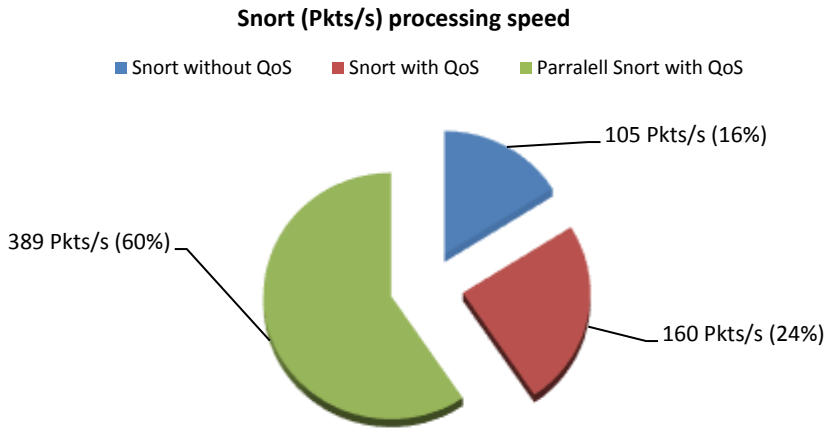


Figure 6. 24: Parallel Snort NIDPS Packets Processing Speed.

6.3.2 Experiment 22: Test NIDPS architecture performance under more than 8 Gbps traffic speed.

In this experiment, TCP replay tool was used to generate traffic at different speeds (Gbps) (see Figure 6.25) through the system. Two 1Gb interfaces were used. Each interface was configured as 4 output queues and connected to a Snort NIDPS node. Each Snort node was configured as 4 Snort instances, one for each output queue of the connected interface. Each queue can reserve buffer up to an interface limit (1 Gb).

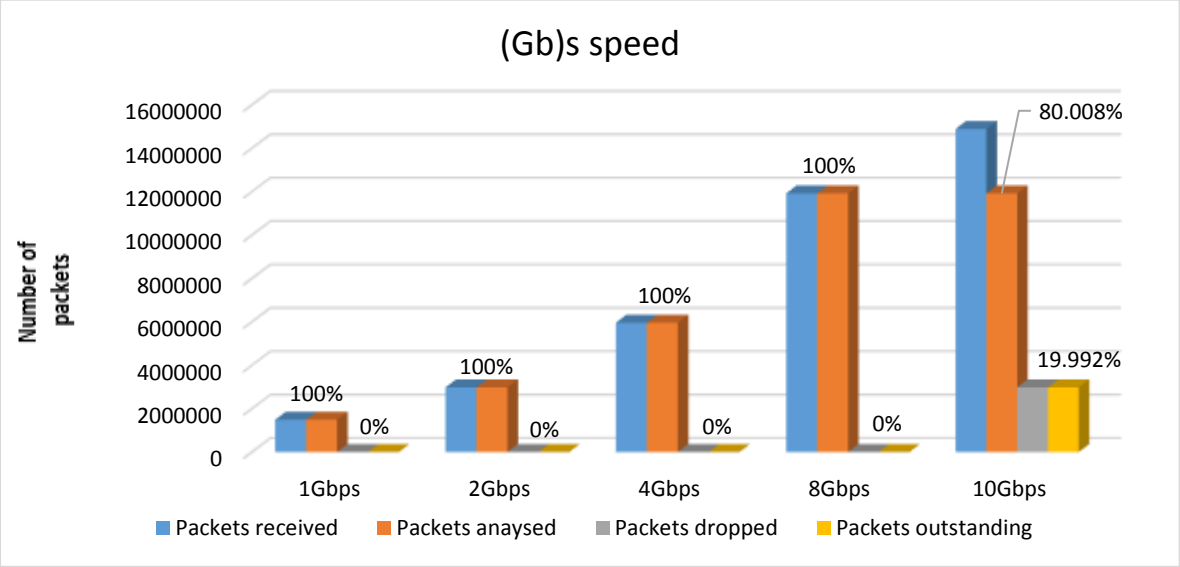


Figure 6. 25: Novel NIDPS architecture with more 8Gbps traffic speed.

As the results show in Figure 6.25, Snort NIDPS processed every single packet that reached the wire. Snort processed 100% of packets that were received while the traffic speed was less than or equal 8 Gbps. When the traffic speed was increased to 10 Gbps , Snort started to drop packets. By using 2x 1Gb interfaces, the experiment results showed that the Snort NIDPS processed more that 8Gbps with 0 packets dropped.

6.4 Summary of experiments

A summary of the results of the stage 2 experiments is shown in Table 6.22

Table 6. 22: A summary of the results

Number of	Purpose of experiment	Snort performance (no QoS no parallel technologies)		QoS technology proposed solution		Section in thesis
		Performance	Packet processing	Performance	Packet processing	

experiment		rate	speed rate (Pkts/sec)	rate	speed rate (Pkts/sec)	
13	Testsolution performance - NIDPS analysis mode (IP traffic)	17%	105	100%	160	6.2.2.1
14	Test solution performance - NIDPS detection mode (IP/ICMP traffic)	50%	94	100%	131	6.2.3.1
15	Test solution performance - NIDPS detection mode (IP/UDP)	7%	76	99%	129	6.2.3.2
16	Test solution performance - NIDPS detection mode (TCP/IP)	100%	17	100%	96	6.2.3.3
17	Test solutionperformance - NIDPS detecting malicious packets	34%	82	99%	857	6.2.3.4
18	Test solution performance - NIDPS using various rules to detect malicious packets	35%	83	99%	853	6.2.4.1
19	Test solution performance - NIDPS prevention mode (IP traffic)	100%	853	100%	4416	6.2.5.1
20	Test performance - NIDPS preventing malicious packets	50%	31	100%	154	6.2.5.2
21	**QoS and Parallel technologies proposed Solution**					6.3.1
	Test solution performance - NIDPS analysis mode with QoS and parallel technologies	**Performance rate**		**Packet processing speed rate (Pkts/sec)**		
		100%		389		
22	Test solution performance - NIDPS architecture with up to 8 Gbps traffic	100%		11904		6.3.2

Experiment 13 has shown how QoS configuration within the Cisco Catalyst 3560 Series switch can enhance performance such that packets are no longer dropped or outstanding; and experiments 14, 15 and 16 show the improvement of NIDPS detection performance with headers such as TCP, UDP and ICMP. Experiments 17 and 18 have shown that how QoS can improve NIDPS performance to detect malicious traffic such that alerts and logs are no longer lost. Experiments 19 and 20 have shown the improvement of NIDPS performance prevention mode with QoS. Experiment 21 shows how parallel technology can be used to speed up packets processing. Finally, experiment 22 shows that NIDPS architecture can process more 8Gbps with 0% dropped.

6.5 Conclusion

NIDPSs are important components for the security of modern computer network systems. An NIDPS needs to perform packet examination of inbound traffic at or near network speed. Otherwise malicious packets may infiltrate the network undetected. This chapter outlined the evaluation results of the novel architecture and unique infrastructure for NIDPS proposed in this thesis. The chapter has presented the results of experiments to show how a novel configuration of QoS and parallel technology can improve NIDPS performance when deployed in a high-speed network. The results show that using the novel QoS configuration in Layer 3 Cisco Catalyst switches with parallel NIDPS significantly improves the NIDPS performance in analysis, detection and prevention mode.

CHAPTER 7: CONCLUSION, RECOMMENDATION AND FUTURE WORK

7.1 Introduction

This chapter summarises the outcomes achieved in the research and then provides recommendations for future research.

7.2 Contribution and achievements

A novel architecture for NIDPS deployment was designed, implemented and evaluated. There has recently been an incredible development in the ways computer networks are used, especially regarding their ability to handle different speeds and data volumes. As a result of this rapid development, computer networks are now more vulnerable than ever to high-speed attacks and threats. These can cause considerable trouble to computer networks and systems. Network intrusions can be categorised at various levels. Many high-speed attacks can be classified as being difficult to detect or prevent. It will become ever more difficult to analyse increasing volumes of traffic due to the rapid shifts in technology that are increasing network speed.

For many years the number of attacks made on networks has been rising dramatically. Network disruptions are often carried out intentionally by several types of direct attack. These attacks are made at various layers in the TCP/IP protocol suite, including the applications layer. Besides the external body, attacks can be made on the network by the internal body as well. Various technologies and methods have been used to prevent such incidents; NIDPS in particular have gained substantial importance.

NIDPS is considered to be one of the best technologies for detecting or preventing threats and attacks. NIDPS closely looks out for any malicious activity in the network and the systems, and reacts to deny or to permit packets from entering the network. An NIDPS is able to provide numerous methods for finding any suspicious packets in normal network traffic. It either directly rejects or blocks suspicious traffic, users, or IP addresses. NIDPSs have attracted the interest of many organisations and governments, and any internet user can deploy them. An NIDPS usually secures a computer system network in four stages: analysis, detection, prevention and correction.

Recently, various open-source tools have become available to cover security requirements for network systems and users. In this research, the performance of an open source NIDPS has been

evaluated in the context of high-speed and volume attacks. The purpose of the evaluation was to determine the performance of the NIDPS under high-speed attack when restricted by off-the-shelf hardware, and then find ways to improve it. 'Snort' was installed and configured as the open source NIDPS. It was chosen for evaluation on account of its being the de facto NIDPS standard. The evaluation system was implemented on real hosts on a real computer network to simulate real-life traffic with malicious packets arriving at different speeds and volumes.

This research focused on the weakness of NIDPS in high-speed network connectivity and proposed a solution for reducing this weakness. It presents a novel architecture in NIDPS development that utilises QoS and parallel technologies to organise and improve network management and traffic processing performance in order to improve the performance of the NIDPS when it is deployed in high-speed traffic.

Many studies are theoretically based and lack supporting practical experimentation. In this research, the author aimed to present practical research in both real and virtual environments, which may be able to move faster to take-up. Furthermore, the author provided intensive technical information as a part of the research that describes the innovative research architecture.

7.2.1 Weaknesses addressed

The current design and implementation of NIDPS was challenged. The most important of the weaknesses found are listed below.

- NIDPS fail to handle high-speed malicious packets and IP traffic.
- The performance rate of Snort NIDPS (analysis, detection and prevention) decreases as the traffic speed increases.
- Snort NIDPS rate of dropped and outstanding packets were very high and increased when traffic speed was increased.
- Snort NIDPS was unable to detect or prevent up to 93% of unwanted IP traffic when the speed was 1KBp0.5ms and up to 65% of malicious packets when malicious packets speed was 255Bp1mSec with flood traffic speed 60KBps. The consequence of this weakness is that standard deployment of Snort NIDPS may not detect or prevent high-speed attacks.

An attack such as flood attack (e.g. ICMP and UDP) uses a high-volume of traffic and a high-speed to create a lot of noise in the network (for example, congestion traffic and buffer overflow) and then reduces system performance, including that of the NIDPS. When packets are outstanding or lost

by the NIDPS, they may traverse the network without any prior analysis. Hence, the network is unprotected against any variety of attack that carries such packets. This fault was addressed by adding other methods to the Snort NIDPS component. Snort NIDPS was unable to analyse a major portion of the headers in a high-speed environment. When traffic was sent at nearly 38,000KBp1ms, Snort analysed less than 17% of the total number of packets that the system received with processing speed 105 Pkts/s. When QoS was used, Snort analysed nearly 99% of the total number of packets that it received in a processing time of 250s with 160 Pkts/s. The incorporation of parallel Snort NIDPS improved its performance by increasing its level of analysis by up to 100% and the speed of system processing by 60% (389 Pkts/s). Snort processor time is reduced from 250s to 102s using three parallel processing nodes.

7.2.2 A Novel Architecture for NIDPS

In order to solve the problems summarised in 7.2.1, a novel QoS configuration using network Layer 3 switch features was designed and implemented in this research. The QoS configuration boosted the NIDPS performance with regard to its congestion management and its congestion avoidance. Congestion management created balanced queuing by evaluating the internal DSCP and determining in which of the four egress queues to place the packets. Other items related to queuing were also configured: defining the priority queue, defining a queue set, guaranteeing buffer availability, limiting memory allocation, specifying buffer allocation, setting drop thresholds, mapping the CoS to the DSCP value, configuring SRR, and limiting the bandwidth on each of the outbound queues. The congestion avoidance method also helped with the performance of the NIDPS, by, for example, setting output queuing, configuring Weighted Tail Drop (WTD) parameters and thresholds for the four-queue set, guaranteeing buffer availability for a queue's maximum memory, and allocating a queue buffer for all the output queues of an interface. The research enhanced policy and classification methods of the standard QoS architecture. The novel QoS architecture showed a substantial improvement in overall network system management, performance and security.

A further important component of the new architecture was the use of parallel NIDPS nodes to match each of the switch egress queues. This enabled NIDPS packet checking to keep up with increased arrival rates typical of an attack. Snort's performance improved markedly, allowing more packets to be checked before they were delivered into the network. The performance (analysis, detection and prevention rate) of Snort NIDPS increased to more than 99%. By using 2 machines (PC) connected to two Gb interfaces, Snort NIDPS processed more than 8 Gbps with 0 drop. This number can be increased up to 32Gbps which is the full system capacity forward bandwidth by implementing more nodes of NIDPS.

7.2.3 Contributions to Knowledge

This research has made the following contributions to knowledge:

(1) A new architecture for NIDPS which uses QoS switch technology and parallel processing nodes has been developed to combat the issue of dropped packets in high-speed signature-based attacks.

Important features of the new architecture are:

- improvement of the NIDPS analysis, detection and prevention rates;
- handling of any high-speed network traffic variations;
- preventing high-speed attacks from inside or outside the network; and an increase of nearly 99 percent in Snort NIDPS performance.

(2) A new NIDPS architecture has been processed for up to 8Gbps traffic speed with 0 packets drop.

There is always a limit to the number of packets a system can receive and process. This may cause serious security breaches. The research adapted an established technique in order to improve the capability of NIDPS, allowing it to deal with any volume of incoming traffic. The technique supports the NIDPS analysis, detection and prevention engine to help it process high-speed traffic. The research focused on establishing technical information of the problem and solution. This information generalises the problem and solution and thus enables the proposed approach to be applied more easily to infrastructures that are different to the testbed used in this research

7.3 Limitations

The output of the research was the design of a new architecture for NIDPS, which is achieved by adding other processing methods to the existing program. The system processed 8Gbps with 0 packets dropped. This number can be improved but it depends on the system capacity which is always limited.

No methodological development was added to Snort except configured multi-node of Snort (parallel component). Snort on its own was found to be vulnerable to the experiments required at diverse stages of the research. Snort never can hold all attacks and malicious packets but should be part of a defence in depth strategy and no single tool or technology should be relied upon exclusively.

To add any hardware, tools or features, a full cost-benefit analysis should be carried out in order to discover whether a sufficient level of security can be achieved at an acceptable cost.

7.4 Further research

The research centred on the failure of NIDPS to adequately handle traffic that occurs in high-speed network attacks. Experiments were carried out which presented the weakness of NIDPSs and which later showed how a novel architecture improved NIDPSs in terms of performance, efficiency and effectiveness. The experimental results show vast improvement in reducing packet loss (drop) and therefore give better protection against attacks. However there remain areas to investigate.

NIDPSs are used to capture data and detect malicious packets that travelling on the network media (cables, wireless) and match them to a database of signatures. Signature-based NIDPS are able to detect known attacks, but the major problem of the signature-based approach is that every signature should have an entry in a database in order to compare with the incoming packets. New signatures arise constantly and an issue is how to keep track up with new signatures. Another problem is processing time required to check all signatures. Knowledge sharing may provide a solution. Cloud computing which provides for massive processing distribution and sharing is a possible future direction but this also raises issues of trust. An avenue of future investigation should aim to develop a trusted cloud solution to NIDPS deployment such that if the threshold monitoring tool indicates that traffic is increasing then extra Snort nodes can be brought into play from the cloud. Future work should investigate the use of specialized and trustworthy security clouds i.e a parallel node of NIDPS can be implemented on a mulit-core/multi-processing cloud environment which can increase the NIDPS processing speed in order to improve its performance.

Statistical based anomaly detection is designed to detect deviations from a baseline model of network behaviour. When the rate of "malicious" packet transmission is very high, the attack will almost certainly be detected by a statistical anomaly detector. Therefore, the fact that Snort's performance falls when the rate of transfers is high might be inconsequential in real world networks. The author considers this issue needs further investigating.

When the rate of malicious packet transmission increases, Snort’s detection power may fall in terms of the True Positive rate (i.e. number of malicious packets detected). However, any anomaly detector faces a trade-off between true detection and network performance. For example, while the True Positive rate may have decreased, the True Negative rate (i.e. proportion of packets that Snort

said were clean and actually were clean) may have increased. As Snort does not provide parameters that allow the network admin to customize this trade-off, more research is needed to justify the trade-off between network performance and security.

In the area of development of the NIDPS detection function, intelligent techniques can be exploited to develop new rules for more precise detection of attacks to counteract the growth in diversity and deviousness. The current and anticipated future demands for online security require the revision of existing systems towards the development of improved parallel systems as well as stronger rule sets. Furthermore, using multi-core processors, further research can be done such as looking into some of the potential technological advancements in NIDPSs that can be employed for beneficial purposes and objectives. Finally, the success of this project has revealed more challenges, as following:

- Importance analysis of interdependencies and possible cascading effects across related processes within the QoS framework.
- Develop and execute a coordinated research to fully utilize the potential of IDPS to capture and analyse attacks trends.
- Generate complex detection, prevention and correction algorithms;

In the system, we identified that there is limitation for the number of packets processing which is 8.0 Gbps with 0% packets dropped. The idea has been examined further in terms of performance limitation above 8.0 Gbps, and therefore modification may be made for better response. As experiment 22 showed, packets started to be dropped when load-balancing for traffic exceeding 8.0 Gbps. Analys is still in development and shall be covered in the future efforts.

Establishing a relationship between traffic size and number of IDPS cluster nodes for an efficient performance is also an interesting research area. Defining parameters to identify the number of nodes for a scalable response to network speed type and will be a good addition.

References

Ahmed, G., Khan, M. N. A., and Bashir, M. S. (2015) 'A Linux-Based IDPS using Snort'. *Computer Fraud & Security* 2015 (8), 13-18.

Akhlaq, M., Alserhani, F., Awan, I., Mellor, J., Cullen, A. J., and Al-Dhelaan, A. (2011) *Implementation and Evaluation of Network Intrusion Detection Systems*. In Network performance engineering (pp. 988-1016). Springer Berlin Heidelberg.

Akhtar, N., Matta, I., and Wang, Y. (2015) 'Managing NFV using SDN and Control Theory'. *CS Department, Boston University, Tech.Rep.BUCS-TR-2015-013*.

Albin, E. and Rowe, N. C. (eds.) (2012) *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*. 'A Realistic Experimental Comparison of the Suricata and Snort Intrusion-Detection Systems': IEEE.

Alder, R., Baker, A.R., Carter, E.F., Esler, J., Foster, J.C., Jonkman, M., Keefer, C., Marty, R. and Seagren, E.S., 2007. Snort: IDS and IPS Toolkit. *Syngress.*

Aldosari, H. M., Snasel, V., and Abraham, A. (2016) 'A Novel Security Layer for Internet of Things.' *Journal of Information Assurance & Security* 11 (2).

Alpcan, T. and Başar, T. (2010) *Network Security: A Decision and Game-Theoretic Approach*. Cambridge University Press.

Al-Saleh, M. I. (ed.) (2015) *The International Technology Management Conference (ITMC2015)*. 'Towards Extending the Antivirus Capability to Scan Network Traffic'.

Alserhani, F., Akhlaq, M., Awan, I. U., Mellor, J., Cullen, A. J., and Mirchandani, P. (eds.) (2009) *5th International Conference on Information Assurance and Security, IAS 2009*. 'Evaluating Intrusion Detection Systems in High Speed Networks'.

Ammons, G. S., Bala, V., Duri, S. S., Mummert, T. W., and Reimer, D. C. (2016). *Passive Monitoring of Virtual Systems using Extensible Indexing*. International Business Machines Corporation. U.S. Patent 9,229,758.

Amudhavel, J., Brindha, V., Anantharaj, B., Karthikeyan, P., Bhuvaneswari, B., Vasanthi, M., Nivetha, D., and Vinodha, D. (2016) 'A Survey on Intrusion Detection System: State of the Art Review'. *Indian Journal of Science and Technology* 9 (11).

Andres, F. A. and Charles, E. B. (2005) 'Snort Diagrams for Developers'. *Universidad del Cauca-Colombia.*

Ansilla, J., Vasudevan, N., JayachandraBensam, J., and Anunciya, J. (eds.) (2015) *Circuit, Power and Computing Technologies (ICCPCT), 2015 International Conference on*. 'Data Security in Smart Grid with Hardware Implementation Against DoS Attacks': IEEE.

AR, M. S. (2015) 'A SOA Security Solution to Prevent Replay Attacks'. *International Journal* 3 (6).

Aydin, M. A., Zaim, A. H., and Ceylan, K. G. (2009) 'A Hybrid Intrusion Detection System Design for Computer Network Security'. *Computers and Electrical Engineering* 35 (3), 517-526.

Balkanli, E. (2015) 'A Comprehensive Study on One-Way Backscatter Traffic Analysis'. *(Master dissertation, Dalhousie University).*

Becchi, M. and Crowley, P. (eds.) (2008) *Proceedings of 2008 ACM CoNEXT Conference - 4th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '08*. 'Extending Finite Automata to Efficiently Match Perl-Compatible Regular Expressions'.

Beg, S., Naru, U., Ashraf, M., and Mohsin, S. (2010) 'Feasibility of Intrusion Detection System with High Performance Computing: A Survey'. *International Journal for Advances in Computer Science* 1 (1).

Bessis, T. C. and Rana, A. V. (2015). *Session Initiation Protocol (SIP) Firewall for IP Multimedia Subsystem (IMS) Core*. U.S. Patent 8,955,090.

Bernstein, R. and Dulkin, A. (2016). *Systems and Methods for Detection of Anomalous Network Behavior.* U.S. Patent 20,160,142,435.

Brahmi, I., Yahia, S. B., and Poncelet, P. (eds.) (2011) *SECRYPT 2011 - Proceedings of the International Conference on Security and Cryptography*. 'A Snort-Based Mobile Agent for a Distributed Intrusion Detection System'.

Bro (2014), *The Bro Network Security Monitor* [online] available from <https://www.bro.org/> [29 July 2016].

Brodle, B. C., Cytron, R. K., and Taylor, D. E. (eds.) (2006) *Proceedings - International Symposium on Computer Architecture*. 'A Scalable Architecture for High-Throughput Regular-Expression Pattern Matching'.

Buchanan, W. J., Flandrin, F., Macfarlane, R., and Graves, J. (2011) 'A Methodology to Evaluate Rate-Based Intrusion Prevention System Against Distributed Denial-of-Service (DDoS).' *Edinburgh Napier University.*

Bulajoul, W., James, A., and Pannu, M. (eds.) (2013) *E-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on*. 'Network Intrusion Detection Systems in High-Speed Traffic in Computer Networks': IEEE

Bul'ajoul, W., James, A., and Pannu, M. (2015) 'Improving Network Intrusion Detection System Performance through Quality of Service Configuration and Parallel Technology'. *Journal of Computer and System Sciences* 81 (6), 981-999

Burton, J. D., Baumrucker, C. T., and Dubrawsky, I. (2003) *Cisco Security Professional's Guide to Secure Intrusion Detection Systems*.: Syngress Publishing

Calvo Moya, M. A. (2008) . *Analysis and Evaluation of the Snort and Bro Network Intrusion Detection Systems*

Caswell, B., Beale, J., and Baker, A. (2007) *Snort Intrusion Detection and Prevention Toolkit*.: Syngress

Caswell, B. and Beale, J. (2004) *Snort 2.1 Intrusion Detection*.: Syngress

Chakrabarti, S., Chakraborty, M., and Mukhopadhyay, I. (eds.) (2010) *ICWET 2010 - International Conference and Workshop on Emerging Trends in Technology 2010, Conference Proceedings*. 'Study of Snort-Based IDS'

Chauhan, K. and Prasad, V. (2015) 'Distributed Denial of Service (DDoS) Attack Techniques and Prevention on Cloud Environment'. *International Journal of Innovations & Advancement in Computer Science, pp.210-215*.

Chen, J., Wong, M., Zhang, L., and Techonlogies, H. (2015) 'Security and Usability'. *User Requirements for Wireless* 42, 77.

Chen, M., Hsiao, Y., Su, H., and Chu, Y. (2015) 'High-Throughput ASIC Design for e-Mail and Web Intrusion Detection'. *IEICE Electronics Express* (0).

Chen, X., Wu, Y., Xu, L., Xue, Y., and Li, J. (2009) 'Para-Snort: A Multi-Thread Snort on Multi-Core Ia Platform'. *Proceedings of Parallel and Distributed Computing and Systems (PDCS)*.

Chen, Y. - and Chen, Y. -. (eds.) (2009) *Proceedings of the ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, CSI-KDD in Conjunction with SIGKDD'09*. 'Combining Incremental Hidden Markov Model and Adaboost Algorithm for Anomaly Intrusion Detection'.

Cheswick, W. R., Bellovin, S. M., and Rubin, A. D. (2003) *Firewalls and Internet Security: Repelling the Wily Hacker*.: Addison-Wesley Longman Publishing Co., Inc.

Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Rowe, J., Staniford-Chen, S., Yip, R., and Zerkle, D. (1999). *The Design of GrIDS: A Graph-Based Intrusion Detection System.*

Chi, R. (2014) 'Intrusion Detection System Based on Snort, 3, 2014, Pp. ' 3 (in: Proceedings of the 9th International Symposium on Linear Drives for Industry Applications, Springer Heidelberg, Berlin), 657-664.

Cisco (2016a) *Catalyst 3560 Switch Software Configuration Guide*. Cisco IOS Release 15.0(2) SE and Later edn. USA: Cisco [online] available from < http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3560/software/release/15-0_2_se/configuration/guide/scg3560.pdf > [31 May 2016].

Cisco (2016b) *Cisco Interfaces and Modules,Cisco Security Modules for Security Appliances* [online] available from <http://www.cisco.com/c/en/us/support/interfaces-modules/security-modules-security-appliances/tsd-products-support-series-home.html> [02 Jun 2016].

Cisco (2014a) *Enterprise QoS Solution Reference Network Design Guide* [online] available from <http://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/WAN_and_MAN/QoS_SRND/QoS-SRND-Book/IPSecQoS.html> [3 May 2015].

Cisco (2014b) *Security Configuration Guide: Access Control Lists, Cisco IOS Release 15SY* [online] available from <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/sec_data_acl/configuration/15-sy/sec-data-acl-15-sy-book.pdf> [06 Dec 2015].

Costa, K. A., Pereira, L. A., Nakamura, R. Y., Pereira, C. R., Papa, J. P., and Falcão, A. X. (2015) 'A Nature-Inspired Approach to Speed Up Optimum-Path Forest Clustering and its Application to Intrusion Detection in Computer Networks'. *Information Sciences* 294, 95-108.

Cui, C., Xue, L., Chiu, C., Kondikoppa, P., and Park, S. (2015) 'Exploring Parallelism and Desynchronization of TCP Over High Speed Networks with Tiny Buffers'. *Computer Communications* 69, 60-68.

Dabir, A. and Matrawy, A. (eds.) (2007) *Innovations in Information Technology, 2007. IIT'07. 4th International Conference on*. 'Bottleneck Analysis of Traffic Monitoring using Wireshark': IEEE

Davis, M. and Morley, J. (2015) 'Phrasal Intertextuality: The Responses of Academics from Different Disciplines to Students' Re-use of Phrases'. *Journal of Second Language Writing* 28, 20-35.

Daxin, T. and Yang, X. (eds.) (2008) *Proceedings - 2008 IFIP International Conference on Network and Parallel Computing, NPC 2008*. 'A Multi-Core Supported Intrusion Detection System'.

De Muila, P. D. and Ferdinand, J. (2010) 'A Novel Intrusion Detection System (Ids) Architecture: Attack Detection Based on Snort for Multistage Attack Scenarios in a Multi-Cores Environment'. *ACM*.

Dhakar, M. and Tiwari, A. (2015) 'The Conceptual and Architectural Design of an Intelligent Intrusion Detection System'. *Improving Information Security Practices through Computational Intelligence*, 100.

Dhillon, N. K. and Ansari, M. U. (2012) 'Enterprise Network Traffic Monitoring, Analysis, and Reporting using Winpcap Tool a Packet Capturing API'. *International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE)* 1 (6), pp: 19-23.

Donaldson, S. E., Siegel, S. G., Williams, C. K., and Aslam, A. (2015) 'Building an Effective Defense'. in *Enterprise Cybersecurity*. ed. by Anon: Springer, 133-156.

Edge, C. and O'Donnell, D. (2016) 'Network Scanning, Intrusion Detection, and Intrusion Prevention Tools'. in *Enterprise Mac Security*. ed. by Anon: Springer, 441-457.

Emmerich, P., Raumer, D., Beifuß, A., Erlacher, L., Wohlfart, F., Runge, T. M., Gallenmuller, S., and Carle, G. (eds.) (2015) *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2015 International Symposium on*. 'Optimizing Latency and CPU Load in Packet Processing Systems': IEEE.

Fossi, M.,etal. (2010) *Symantec Global Internet Security Threat Report 2009*: Symantec enterprise security.

Fortunati, S., Gini, F., Greco, M. S., Farina, A., Graziano, A., and Giompapa, S. (2016) 'An Improvement of the State-of-the-Art Covariance-Based Methods for Statistical Anomaly Detection Algorithms'. *Signal, Image and Video Processing* 10 (4), 687-694.

Fraize, J., Covell, D., Williams, T., and Tomanek, S. (2015). *Network Attack Offensive Appliance*

Fraser, N. J., Moss, D. J., Lee, J., Tridgell, S., Jin, C. T., and Leong, P. H. (eds.) (2015) *Field Programmable Logic and Applications (FPL), 2015 25th International Conference on*. 'A Fully Pipelined Kernel Normalised Least Mean Squares Processor for Accelerated Parameter Optimisation': IEEE.

Fredj, O. B. (2015) 'A Realistic graph-based Alert Correlation System'. *Security and Communication Networks*.

Fuchsberger, A. (2005) 'Intrusion Detection Systems and Intrusion Prevention Systems,'. *Information Security Group, Royal Holloway, University of London, Egham, Surrey TW20*.

Gallagher, S. (2012) 'Bad Bots: DDoS Attacks Spike in First Quarter, Outdoing all of 2011'. *Ars Technical* (April, 11).

Gamer, T. (2012) 'Collaborative Anomaly-Based Detection of Large-Scale Internet Attacks'. *Computer Networks* 56 (1), 169-185.

Gamer, T. (ed.) (2009) *GLOBECOM - IEEE Global Telecommunications Conference*. 'Anomaly-Based Identification of Large-Scale Attacks'.

Gamer, T. (ed.) (2008) *Proceedings of 2008 ACM CoNEXT Conference - 4th International Conference on Emerging Networking EXperiments and Technologies, CoNEXT '08*. 'Distributed Detection of Large-Scale Attacks in the Internet'.

Gao, J. and Xiao, Y. (2012) 'Design for Accountability in Multi-Core Networks'. *Journal of Convergence* 3 (3), 9-16.

García-Teodoro, P., Díaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. (2009) 'Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges'. *Computers and Security* 28 (1-2), 18-28.

Genge, B., Graur, F., and Haller, P. (2015) 'Experimental Assessment of Network Design Approaches for Protecting Industrial Control Systems'. *International Journal of Critical Infrastructure Protection* 11, 24-38.

Goel, J. N. and Mehtre, B. (eds.) (2016) *Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics*. 'Stack Overflow Based Defense for IPv6 Router Advertisement Flooding (DoS) Attack': Springer.

Gold, S. (2011) 'The Future of the Firewall'. *Network Security* 2011 (2), 13-15.

Gullett, D. (2012) 'Snort 2.9. 3 and Snort Report 1.3. 3 on Ubuntu 12.04 Lts Installation Guide'.

Gupta, S. (2012). *A Graphical User Interface Framework for Detecting Intrusions using Bro IDS*. (Doctoral dissertation, Thapap University Patiala).

Hanani, A., Carey, M. J., and Russell, M. J. (2012) 'Language Identification using Multi-Core Processors'. *Computer Speech and Language* 26 (5), 371-383.

Hernandez-Herrero, J. and Solworth, J. A. (2007) 'The Need for a Multi-Perspective Approach to Solve the DDoS Problem'. *Bell Labs Technical Journal* 12 (3), 121-130.

Hofstede, R. and Pras, A. (2012) *Real-Time and Resilient Intrusion Detection: A Flow-Based Approach*.

Hoque, M. S., Mukit, M., Bikas, M., and Naser, A. (2012) 'An Implementation of Intrusion Detection System using Genetic Algorithm'. *ArXiv Preprint arXiv:1204.1336*.

Hsu, C. -. and Wang, S. -. (eds.) (2011) *Proceedings of the International Conference on Parallel Processing Workshops*. 'Embedded Network Intrusion Detection Systems with a Multi-Core Aware Packet Capture Module'.

Hussain, J., Lalmuanawma, S., and Chhakchhuak, L. (2015) 'A Novel Network Intrusion Detection System using Two-Stage Hybrid Classification Technique'. *Ijccer* 3 (2), 16-27.

Hussein, S. M., Ali, F. H. M., and Kasiran, Z. (eds.) (2012) *2012 2nd International Conference on Digital Information and Communication Technology and its Applications, DICTAP 2012*. 'Evaluation Effectiveness of Hybrid IDS using Snort with Naïve Bayes to Detect Attacks'.

Irland, M. I. (1978) 'Buffer Management in a Packet Switch'. *Communications, IEEE Transactions on* 26 (3), 328-337.

Jaiswal, R., Lokhande, S., and Gulavani, A. 'Implementation and Analysis of DoS Attack Detection Algorithms'.

Jamshed, M. A., Lee, J., Moon, S., Yun, I., Kim, D., Lee, S., Yi, Y., and Park, K. (eds.) (2012) *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. 'Kargus: A Highly-Scalable Software-Based Intrusion Detection System': ACM.

Jang-Jaccard, J. and Nepal, S. (2014) 'A Survey of Emerging Threats in Cybersecurity'. *Journal of Computer and System Sciences* 80 (5), 973-993.

Je, H., Kwon, D., and Ju, H. (eds.) (2015) *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. 'Design of a Media Stream Relay Engine on the Android OS': IEEE.

Jiang, H., Zhang, G., Xie, G., Salamatian, K., and Mathy, L. (eds.) (2013) *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. 'Scalable High-Performance Parallel Design for Network Intrusion Detection Systems on Many-Core Processors': IEEE Press.

Jiang, W., Song, H., and Dai, Y. (2005) 'Real-Time Intrusion Detection for High-Speed Networks'. *Computers & Security* 24 (4), 287-294.

Jiang, H., Yang, J., and Xie, G. (eds.) (2011) *Proc. 10th IEEE Int. Conf. on Trust, Security and Privacy in Computing and Communications, TrustCom 2011, 8th IEEE Int. Conf. on Embedded Software and Systems, ICESS 2011, 6th Int. Conf. on FCST 2011*. 'Exploring and Enhancing the Performance of Parallel IDS on Multi-Core Processors'.

Jiang, W., Song, H., and Dai, Y. (2005) 'Real-Time Intrusion Detection for High-Speed Networks'. *Computers and Security* 24 (4), 287-294.

Jones, J. K. and Romney, G. W. (eds.) (2004) *SIGITE 2004 Conference*. 'Honeynets: An Educational Resource for IT Security'.

Karthikeyan, K. and Indra, A. (2010) 'Intrusion Detection Tools and techniques—a Survey'. *International Journal of Computer Theory and Engineering* 2 (6), 901-906.

Kawahara, J., Kobayashi, K. M., and Maeda, T. (2015) 'Tight Analysis of Priority Queuing for Egress Traffic'. *Computer Networks* 91, 614-624.

Kelly, T. (2003) 'Scalable TCP: Improving Performance in Highspeed Wide Area Networks'. *ACM SIGCOMM Computer Communication Review* 33 (2), 83-91.

Kenkre, P. S., Pai, A., and Colaco, L. (eds.) (2015) *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*. 'Real Time Intrusion Detection and Prevention System': Springer.

Kerner, S.,M. (2012) 'HP Report: More Attacks, Despite Fewer New Vulnerabilities overall '. *E-Security Planet* (April, 19).

Kesselman, A., Patt-Shamir, B., and Scalosub, G. (2013) 'Competitive Buffer Management with Packet Dependencies'. *Theoretical Computer Science* 489, 75-87.

Khamphakdee, N., Benjamas, N., and Saiyod, S. (2015) 'Improving Intrusion Detection System Based on Snort Rules for Network Probe Attacks Detection with Association Rules Technique of Data Mining'. *Journal of ICT Research and Applications* 8 (3), 234-250.

Khamphakdee, N., Benjamas, N., and Saiyod, S. (eds.) (2014) *Information and Communication Technology (ICoICT), 2014 2nd International Conference on*. 'Improving Intrusion Detection System Based on Snort Rules for Network Probe Attack Detection': IEEE.

Khorchani, B., Halle, S., and Villemaire, R. (eds.) (2012) *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*. 'Firewall Anomaly Detection with a Model Checker for Visibility Logic'.

Kim, H., Pamnami, A., and Patel, M. (2007) 'State-of-the-Art in Intrusion Detection Systems'. *Dept.of Electrical and Computer Engineering Stevens Institute of Technology, Hoboken, New Jersey*.

Kim, J., Kim, A., Yuk, J., and Jung, H. (2015) 'A Study on Wireless Intrusion Prevention System Based on Snort'. *International Journal of Software Engineering and its Applications* 9 (2), 1-12.

Kim, I. and Cho, Y. (2012) *A Security Framework for Blocking New Types of Internet Worms in Ubiquitous Computing Environments*.

Kishore, K. R., Hendel, A., and Kalkunte, M. V. (2015). *System, Method and Apparatus for Network Congestion Management and Network Resource Isolation.*

Kopek, C. V., Fulp, E. W., and Wheeler, P. S. (eds.) (2007) *Proceedings - IEEE Military Communications Conference MILCOM*. 'Distributed Data Parallel Techniques for Content-Matching Intrusion Detection Systems'.

Kouvastos, D. (2011) 'Network Performance Engineering: A Handbook on Convergent Multi-Service Networks and Next Genereation Internet'. *German: Springer-Verlag Berlin Heidelberg*.

Kozushko, H. (2003) 'Intrusion Detection: Host-Based and Network- Based Intrusion Detection Systems'. *Independent Study*.

KR, K. and Indra, A. (2010) 'Intrusion Detection Tools and techniques–a Survey'. *International Journal of Computer Theory and Engineering, 2(6), p.901*.

Kreibich, C. and Crowcroft, J. (eds.) (2004) *Computer Communication Review*. 'Honeycomb - Creating Intrusion Detection Signatures using Honeypots'.

Kruegel, C. V., G. (2003) 'Anomaly Detection of Web-Based Attacks' (October 2003).*ACM*.

Kruegel, C., Vigna, G., and Robertson, W. (2005) 'A Multi-Model Approach to the Detection of Web-Based Attacks'. *Computer Networks* 48 (5), 717-738.

Kruegel, C. and Vigna, G. (eds.) (2003) *Proceedings of the ACM Conference on Computer and Communications Security*. 'Anomaly Detection of Web-Based Attacks'.

Lee, J., Chang, K., Jun, C., Cho, R., Chung, H., and Lee, H. (2015) 'Kernel-Based Calibration Methods Combined with Multivariate Feature Selection to Improve Accuracy of Near-Infrared Spectroscopic Analysis'. *Chemometrics and Intelligent Laboratory Systems* 147, 139-146.

Lee, J., Moskovics, S., and Silacci, L. (1999) 'A Survey of Intrusion Detection Analysis Methods'. *Cse* 221.

Lennon, M. (2012) 'Report: DDoS Attacks Against Banks Spike in Q1 2012, Attack Duration Declines'. *Security Week* (April, 12).

Lerace, N., Urrutia, C., and Basset, R. (2005) 'Intrusion Prevention Systems'. *Ubiquity*, v6-i19.

Li, M., Deng, J., Liu, L., Long, Y., and Shen, Z. (2015) 'Evacuation Simulation and Evaluation of Different Scenarios Based on Traffic Grid Model and High Performance Computing'. *International Review for Spatial Planning and Sustainable Development* 3 (3), 4-15.

Li, Z., Gao, Y., and Chen, Y. (2010) 'HiFIND: A High-Speed Flow-Level Intrusion Detection Approach with DoS Resiliency'. *Computer Networks* 54 (8), 1282-1299.

Li, Z., Gao, Y., and Chen, Y. (2010) 'HiFIND: A High-Speed Flow-Level Intrusion Detection Approach with DoS Resiliency'. *Computer Networks* 54 (8), 1282-1299.

Liew, K.M., Shen, H., See, S., Cai, W., Fan, P. and Horiguchi, S. eds., 2004.*Parallel and Distributed Computing: Applications and Technologies: 5th International Conference, PDCAT 2004, Singapore, December 8-10, 2004, Proceedings* (Vol. 3320). Springer.

Limmer, T. and Dressler, F. (eds.) (2011) *Proceedings - International Conference on Computer Communications and Networks, ICCCN*. 'Adaptive Load Balancing for Parallel IDS on Multi-Core Systems using Prioritized Flows'.

Limmer, T. and Dressler, F. (eds.) (2011) *2011 IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2011*. 'Improving the Performance of Intrusion Detection using Dialog-Based Payload Aggregation'.

Liu, J. -. and Dong, F. -. (eds.) (2008) *Proceedings - International Symposium on Computer Science and Computational Technology, ISCSCT 2008*. 'A Dynamic Adaptive Load Balance Algorithm in Parallel Intrusion Detection System'.

Liu, T., Wang, Z., Wang, H., and Lu, K. (2012) 'An Entropy-Based Method for Attack Detection in Large Scale Network'. *International Journal of Computers, Communications and Control* 7 (3), 509-517.

Lutz, T., Fensch, C., and Cole, M. (eds.) (2015) *Proceedings of the 8th Workshop on General Purpose Processing using GPUs*. 'Helium: A Transparent Inter-Kernel Optimizer for OpenCL': ACM.

Makanju, A., LaRoche, P., and Zincir-Heywood, A. N. (eds.) (2007) *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Security and Defense Applications, CISDA 2007*.

'A Comparison between Signature and GP-Based IDSs for Link Layer Attacks on WiFi Networks'.

Malik, M. and Singh, Y. (2015) 'A Review: DoS and DDoS Attacks'. *International Journal of Computer Science and Mobile Computing, Vol.4 Issue.6, June- 2015, pg. 260-265.*

Mantur, B., Desai, A., and Nagegowda, K. (2015) 'Centralized Control Signature-Based Firewall and Statistical-Based Network Intrusion Detection System (NIDS) in Software Defined Networks (SDN)'. in *Emerging Research in Computing, Information, Communication and Applications*. ed. by Anon: Springer, 497-506.

Marinova-Boncheva, V. (2007) 'A Short Survey of Intrusion Detection Systems'. *Problems of Engineering Cybernetics and Robotics.*

Mehra, P. (2012) 'A Brief Study and Comparison of Snort and Bro Open Source Network Intrusion Detection Systems'. *International Journal of Advanced Research in Computer and Communication Engineering* 1 (6), 383-386.

Menga, J. (2003) *CCNP Practical Studies: SwitchingCCNP Self-StudyCisco Press Practical Studies Series*. illustrated edn. US: Cisco Press.

Mitra, A., Najjar, W., and Bhuyan, L. (eds.) (2007) *ANCS'07 - Proceedings of the 2007 ACM Symposium on Architecture for Networking and Communications*. 'Compiling PCRE to FPGA for Accelerating SNORT IDS'.

Mudzingwa, D. and Agrawal, R. (eds.) (2012) *Conference Proceedings - IEEE SOUTHEASTCON*. 'A Study of Methodologies used in Intrusion Detection and Prevention Systems (IDPS)'.

Mutz, D., Valeur, F., Vigna, G., and Kruegel, C. (2006) 'Anomalous System Call Detection'. *ACM Transactions on Information and System Security* 9 (1), 61-93.

Naouri, Y. and Perlman, R. (2015). *Network Congestion Management by Packet Circulation*. U.S. Patent 8,989,017.

Naveen, N., Natarajan, S., and Srinivasan, R. (eds.) (2012) *Advanced Computer Science Applications and Technologies (ACSAT), 2012 International Conference on*. 'Application of Change Point Outlier Detection Methods in Real Time Intrusion Detection': IEEE.

Nazer, G. M. and Selvakumar, A. A. L. (2011) 'Current Intrusion Detection Techniques in Information Technology - A Detailed Analysis'. *European Journal of Scientific Research* 65 (4), 611-624.

NetScanToolsPro (2013) *NetScanToolsPro Technical Info* [online] available from <Available .http://www.netscantools.com/index.html> [28 July 2015].

Niu, S., Mo, J., Zhang, Z., and Lv, Z. (eds.) (2014) *2nd International Conference on Soft Computing in Information Communication Technology*. 'Overview of Linux Vulnerabilities': Atlantis Press.

Ordi, A., Zamani, M., Idris, N. B., Manaf, A. A., and Abdullah, M. S. (2015) 'A Novel WLAN Client Puzzle Against DoS Attack Based on Pattern Matching'. *Mathematical Problems in Engineering* 501, 707548.

Ormazabal, G. S., Schulzrinne, H. G., Yardeni, E., and Patnaik, S. B. (2015) . *Prevention of Denial of Service (DoS) Attacks on Session Initiation Protocol (SIP)-Based Systems using Return Routability Check Filtering.*

Özçelik, İ. and Brooks, R. R. (2015) 'Deceiving Entropy Based DoS Detection'. *Computers & Security* 48, 234-245.

Pachghare, V., Kulkarni, P., and Nikam, D. (2009) 'Overview of Intrusion Detection Systems'. *International Journal of Computer Science and Engineering Systems* 3 (3), 265-268.

Pal, K. and Verma, J. S. (2015) 'A Survey on Anomaly Based Malware Detection and Demolition in False Alarm Rate'.

Parker, D., B. (1981) *Computer Security Management*. United States: Reston Publishing Co, Inc.

Patcha, A. and Park, J. -. (2007) 'An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends'. *Computer Networks* 51 (12), 3448-3470.

Patel, A., Taghavi, M., Bakhtiyari, K., and Júnior, J. C. (2013) 'An Intrusion Detection and Prevention System in Cloud Computing: A Systematic Review'. *Journal of Network and Computer Applications* 36 (1), 25-41.

Paxson, V. (2007) 'Bro Intrusion Detection System Hands-on Workshop: Bro Overwiew'.

Podofillini,L.,Sudret,B.,Stojadinovic,B.,Zio,E.,and Kröger, W. (ed.) (2015) *Safety and Reliability of Complex Engineered Systems.* London, UK: CRC Press.

Proudfoot, R., Kent, K., Aubanel, E., and Chen, N. (eds.) (2008) *Proceedings the 19th IEEE/IFIP International Symposium on Rapid System Prototyping - Shortening the Path from Specification to Prototype, RSP 2008*. 'Flexible Software-Hardware Network Intrusion Detection System'.

Qia, Q. and Wang, Z. (2012) 'A New Attack Detection in Large Scale Network Based on Entropy'. *Journal of Networks* 7 (5), 863-868.

Radhakishan, V. and Selvakumar, S. (eds.) (2011) *Proceedings - 2nd International Conference on Networking and Distributed Computing, ICNDC 2011*. 'Prevention of Man-in-the-Middle Attacks using ID Based Signatures'.

Rahman, R. U. (2003) 'Intrusion Detection System with Snort:Advanced IDS Techniques with Snort'. *Apache, PHP, MySQL and ACID: Pearson Education.*

Rajeswari, G. and Nithya, B. (eds.) (2009) *ARTCom 2009 - International Conference on Advances in Recent Technologies in Communication and Computing*. 'Implementing Intrusion Detection System for Multicore Processor'.

Ramraj, E. and Rajan, A. S. (eds.) (2009) *2009 International Conference on Signal Processing Systems, ICSPS 2009*. 'Using Multi-Core Processor to Support Network Parallel Image Processing Applications'.

Rastogi, V., Shao, R., Chen, Y., Pan, X., Zou, S., and Riley, R. (2016) 'Are these Ads Safe: Detecting Hidden Attacks through the Mobile App-Web Interfaces'

Rehman, R. U. (2003) *Intrusion Detection Systems with Snort: Advanced IDS Techniques using Snort, Apache, MySQL, PHP, and ACID*.: Prentice Hall Professional.

Ren, X. and Dong, Z. (2003) 'Snort Rules to Improve Methods of Matching the Speed of Research and the Realization of'. *Computer Application* 23 (4), 59-61.

Roozbahani, A. R. and Rikhtechi, L. (eds.) (2010) *2010 the 2nd International Conference on Computer and Automation Engineering, ICCAE 2010*. 'Creating a Collaborative Architecture in Snorts to High Speed Networks'.

Roozbahani, A. R., Nassiri, R., and Latif-Shabgahi, G. (eds.) (2009) *IFCSTA 2009 Proceedings - 2009 International Forum on Computer Science-Technology and Applications*. 'Attacks Classification to Improve the Power of Snorts'.

Rosebrock, E. and Filson, E. (2006) *Setting Up LAMP: Getting Linux, Apache, MySQL, and PHP Working Together*.: John Wiley & Sons.

Ru, X., Liu, Z., Huang, Z., and Jiang, W. (2016) 'Normalized Residual-Based Constant False-Alarm Rate Outlier Detection'. *Pattern Recognition Letters* 69, 1-7.

Ryu, S. H., Casati, F., Skogsrud, H., Benatallah, B., and Saint-Paul, R. (2008) 'Supporting the Dynamic Evolution of Web Service Protocols in Service-Oriented Architectures'. *ACM Transactions on the Web* 2 (2).

Saboor, A., Akhlaq, M., and Aslam, B. (eds.) (2013) *Information Assurance (NCIA), 2013 2nd National Conference on*. 'Experimental Evaluation of Snort Against DDoS Attacks Under Different Hardware Configurations': IEEE.

Sadhukhan, K., Mallari, R. A., and Yadav, T. (eds.) (2015) *Computing and Network Communications (CoCoNet), 2015 International Conference on*. 'Cyber Attack Thread: A Control-Flow Based Approach to Deconstruct and Mitigate Cyber Threats': IEEE.

Saeed, A., Ahmadinia, A., and Just, M. (2016) 'Secure on-Chip Communication Architecture for Reconfigurable Multi-Core Systems'. *Journal of Circuits, Systems and Computers*, 1650089

Salah, K. and Qahtan, A. (2009) 'Implementation and Experimental Performance Evaluation of a Hybrid Interrupt-Handling Scheme'. *Computer Communications* 32 (1), 179-188.

Salah, K. and Qahtan, A. (2009) 'Implementation and Experimental Performance Evaluation of a Hybrid Interrupt-Handling Scheme'. *Computer Communications* 32 (1), 179-188.

Salah, K. and Kahtani, A. (2009) 'Improving Snort Performance Under Linux'. *IET Communications* 3 (12), 1883-1895.

Samani, M. D. and Karamta, M. (2016) 'Intrusion Detection System for DoS Attack in Cloud'. *International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868 Foundation of Computer Science FCS, New York, USA Volume 10 – No.5*.

Scarfone, K. and Mell, P. (2007) 'Guide to Intrusion Detection and Prevention Systems (Idps)'. *NIST Special Publication* 800 (2007), 94.

Schneider, F., Wallerich, J., and Feldmann, A. (2007) 'Packet Capture in 10-Gigabit Ethernet Environments using Contemporary Commodity Hardware'. in *Passive and Active Network Measurement*. ed. by Anon: Springer, 207-217.

Schneider, J. P. (2012). *Tunneling Ssl Over Ssh*. U.S. Patent 8,190,771.

Schuff, D. L., Choe, Y. R., and Pai, V. S. (eds.) (2008) *Performance Analysis of Systems and Software, 2008. ISPASS 2008. IEEE International Symposium on*. 'Conservative Vs. Optimistic Parallelization of Stateful Network Intrusion Detection': IEEE.

Shaffali Gupta, S. K. (2013) *A GUI Framework for Detecting Intrusions using Bro IDS*. UK: LAP Lambert Academic.

Shirazi, H. M. (2009) 'Anomaly Intrusion Detection System using Information Theory, K-NN and KMC Algorithms'. *Australian Journal of Basic and Applied Sciences* 3 (3), 2581-2597.

Shiri, F. I., Shanmugam, B., and Idris, N. B. (eds.) (2011) *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on*. 'A Parallel Technique for Improving the Performance of Signature-Based Network Intrusion Detection System': IEEE.

Shuo, L. and Quan, Z. (2015) 'Research of Intrusion Protection System using Correlation Policy'. *International Conference on Materials Engineering and Information Technology Applications (MEITA 2015).*

Smith, S. C., Hammell, R. J., Parker, T. W., and Marvel, L. M. (eds.) (2014) *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on*. 'A Theoretical Exploration of the Impact of Packet Loss on Network Intrusion Detection': IEEE.

Snort (2016) *The Snort Documents* [online] available from <https://www.snort.org/documents> [24 July 2016].

Sobh, T. S. (2006) 'Wired and Wireless Intrusion Detection System: Classifications, Good Characteristics and State-of-the-Art'. *Computer Standards and Interfaces* 28 (6), 670-694.

Sommer, R. and Paxson, V. (eds.) (2003) *Proceedings of the ACM Conference on Computer and Communications Security*. 'Enhancing Byte-Level Network Intrusion Detection Signatures with Context'.

Souissi, S., Sliman, L., and Charroux, B. (2016) 'A Novel Security Architecture Based on Multi-Level Rule Expression Language'. in *Hybrid Intelligent Systems*. ed. by Anon: Springer, 259-269.

Stanciu, N. (2013) 'Technologies, Methodologies and Challenges in Network Intrusion Detection and Prevention Systems'. *Informatica Economica* 17 (1), 144.

Staniford-Chen, S., Cheung, S., Crawford ,R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., and Zerkle, D. (1996) 'GrIDS - A Graph-Based Intrusion Detection System for Large Networks '. *Citeseex.*

Stocks, D.(nd) 'Beginner’s Guide to Securing IPv6'. *East Carolina University.*

Subramanian, N. and Rao, S. (eds.) (2009) *2009 1st International Conference on Computational Intelligence, Communication Systems and Networks, CICSYN 2009*. 'Content-Split Based Effective String-Matching for Multi-Core Based Intrusion Detection Systems'.

Szigeti, T., Hattingh, C., Barton, R., and Briley, K. (2013) *End-to-End QoS Network Design: Quality of Service for Rich-Media and Cloud Networks*.: Pearson Education.

Szigeti, T. and Hattingh, C. (2004) 'Quality of Service Design Overview'. *Cisco, San Jose, CA, Dec.*

Tabia, K., Benferhat, S., and Djouadi, Y. (eds.) (2008) *Proceedings - Conference on Local Computer Networks, LCN*. 'A Two-Stage aggregation/thresholding Scheme for Multi-Model Anomaly-Based Approaches'.

Teodoro, G., Kurc, T., Andrade, G., Kong, J., Ferreira, R., and Saltz, J. (2015) 'Performance Analysis and Efficient Execution on Systems with Multi-Core CPUs, GPUs and MICs'. *ArXiv Preprint arXiv:1505.03819* .

Tessel, J, D., Young, S. and Linder, F (2044) *The Hacker's Handbook: The Strategy Behind Breaking into and Defending*. New York: Auerbach Publications.

Thanasekaran, R. D. (2011) 'A Robust and Efficient Real Time Network Intrusion Detection System using Artificial Neural Network in Data Mining'. *International Journal of Information Technology Convergence and Services (IJITCS) Vol* 1.

The Hacker News (2016) *Security in a serious way* [online] available from <http://thehackernews.com/2016/01/biggest-ddos-attack.html> [08 Jan 2016]

Tian, D., Liu, Y., and Xiang, Y. (2009) 'Large-Scale Network Intrusion Detection Based on Distributed Learning Algorithm'. *International Journal of Information Security* 8 (1), 25-35.

Tiwari, R. and Jain, A. (eds.) (2012) *ACM International Conference Proceeding Series*. 'Improving Network Security and Design using Honeypots'.

Topallar, M. (2009) *A New Host-Based Hybrid IDS Architecture - A Mind of its Own: The Know-how of Host-Based Hybrid Intrusion Detection System Architecture using Machine Learning Algorithms with Feature Selection*. Germany ©2009 .: VDM Verlag Saarbrücken, Germany.

Trabelsi, Z., Sayed, H. E., and Zeidan, S. (eds.) (2012) *3rd International Conference on Communications and Networking, ComNet 2012*. 'Firewall Packet Matching Optimization using Network Traffic Behavior and Packet Matching Statistics'.

Trevisan, M., Finamore, A., Mellia, M., Munafò, M., and Rossi, D. (2016) 'DPDKStat: 40Gbps Statistical Traffic Analysis with Off-the-Shelf Hardware'.

Valdes, A. and Skinner, K. (2001) 'Probabilistic Alert Correlation'. *Proc of the 4th International Symposium, Recent Advances in Intrusion Detection (RAID).*

Vasanthi, S. and Chandrasekar, S. (eds.) (2011) *IET Seminar Digest*. 'A Study on Network Intrusion Detection and Prevention System Current Status and Challenging Issues'.

Vasiliadis, G., Polychronakis, M., and Ioannidis, S. (eds.) (2011) *Proceedings of the 18th ACM Conference on Computer and Communications Security*. 'MIDeA: A Multi-Parallel Intrusion Detection Architecture': ACM.

Vermeiren, T., Borghs, E., and Haagdorens, B. (eds.) (2004) *IEEE Consumer Communications and Networking Conference, CCNC*. 'Evaluation of Software Techniques for Parallel Packet Processing on Multi-Core Processors'.

Vigna, G. and Kruegel, C. (2006) *Host-Based Intrusion Detection*.: na.

Wallace, K. (2011) *Implementing Cisco Unified Communications Voice Over IP and QoS (CVOICE) Foundation Learning Guide:(CCNP Voice CVoice 642-437)*.: Cisco Press.

Wang, B., Zheng, Y., Lou, W., and Hou, Y. T. (2015) 'DDoS Attack Protection in the Era of Cloud Computing and Software-Defined Networking'. *Computer Networks* 81, 308-319.

Wang, J. and Kissel, Z. A. (2015) 'Introduction to Network Security: Theory and Practice'. *John Wiley & Sons*.

Wang, K. (ed.) (2004) *Talk at the Fifth HOPE Conference*. 'Frustrating OS Fingerprinting with Morph'.

Wang, C., Zhang, Z., and Song, X. (2012) *Research on the Information Security Technology of University Campus Network*.

Wang, W. and Zhang, X. (eds.) (2011) *Proceedings of the ACM Symposium on Applied Computing*. 'High-Speed Web Attack Detection through Extracting Exemplars from HTTP Traffic'.

Wang, X., Qi, Y., Yang, B., Xue, Y., and Li, J. (eds.) (2009) *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*. 'Towards High-Performance Network Intrusion Prevention System on Multi-Core Network Services Processor'.

Wang, Y., Xiang, Y., Zhang, J., Zhou, W. and Xie, B., 2014. Internet traffic clustering with side information. *Journal of Computer and System Sciences*, 80(5), pp.1021-1036.

Weaver, R., Weaver, D., and Farwood, D. (2013) *Guide to Network Defense and Countermeasures*.: Cengage Learning.

Wheeler, P. and Fulp, E. (eds.) (2007) *Proceedings of the 45th Annual Southeast Regional Conference*. 'A Taxonomy of Parallel Techniques for Intrusion Detection': ACM.

Whitman, M., Mattord, H., Mackey, D., and Green, A. (2012) *Guide to Network Security*.: Cengage Learning.

WinPcap (2013) *WinPcap, the Industry-Standard Windows Packets Capture Library* [online] available from <http://www.winpcap.org> [05/11 2015].

Wu, H., Cadambi, S., and Chakradhar, S. T. (2015). *Optimizing Data Warehousing Applications for GPUs using Dynamic Stream Scheduling and Dispatch of Fused and Split Kernels.*

Wu, H., Schwab, S., and Peckham, R. L. (2008). *Signature Based Network Intrusion Detection System and Method.*

Wu, T., M. (2009) 'Intrusion Detection Systems, Sixth Ed., Intrusion Assurance Technology Analysis Center '. *(Herndon, VA,).*

Wun, B., Crowley, P., and Raghunth, A. (eds.) (2009) *ANCS'09: Symposium on Architecture for Networking and Communications Systems*. 'Parallelization of Snort on a Multi-Core Platform'.

Xi, J. (ed.) (2011) *Proceedings - 2011 7th International Conference on Computational Intelligence and Security, CIS 2011*. 'A Design and Implement of IPS Based on Snort'.

Xiang, Y. and Zhou, W. (eds.) (2008) *Proceedings of the IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*. 'Using Multi-Core Processors to Support Network Security Applications'.

Yan, F., Jian-Wen, Y., and Lin, C. (eds.) (2015) *Measuring Technology and Mechatronics Automation (ICMTMA), 2015 Seventh International Conference on*. 'Computer Network Security and Technology Research': IEEE.

Yang, J. -. and Lu, J. -. (eds.) (2011) *Proceedings 2011 International Conference on Mechatronic Science, Electric Engineering and Computer, MEC 2011*. 'Linked IPS on Octeon Multi-Core Processor for New Generation Network'.

Yang, W., Fang, B. -., Liu, B., and Zhang, H. -. (2004) 'Intrusion Detection System for High-Speed Network'. *Computer Communications* 27 (13), 1288-1294.

Zhang, L., Shen, P., Dong, L., Song, J., Feng, Y., Liu, Q., Yi, K., Zhi, L., and Zhao, W. (2015) 'Research and Design of Network Intrusion Detection System Based on Multi-Core Tilera64'. *Information Technology and Applications*, 371.

Zhang, J. and Zulkernine, M. (eds.) (2006) *Proceedings - First International Conference on Availability, Reliability and Security, ARES 2006*. 'A Hybrid Network Intrusion Detection Technique using Random Forests'.

Zhao, J., Wang, L., Tao, J., Chen, J., Sun, W., Ranjan, R., Kołodziej, J., Streit, A., and Georgakopoulos, D. (2014) 'A Security Framework in G-Hadoop for Big Data Computing Across Distributed Cloud Data Centres'. *Journal of Computer and System Sciences* 80 (5), 994-1007.

Zhao, X., Jiang, Y., and Stathaki, T. (2016) 'A Novel Low False Alarm Rate Pedestrian Detection Framework Based on Single Depth Images'. *Image and Vision Computing* 45, 11-21.

Zhao, K., Chu, J., Che, X., Lin, L., and Hu, L. (eds.) (2008) *2nd International Conference on Anti-Counterfeiting, Security and Identification, ASID 2008*. 'Improvement on Rules Matching Algorithm of Snort Based on Dynamic Adjustment'.

Zhao, X. -. and Sun, J. -. (eds.) (2003) *International Conference on Machine Learning and Cybernetics*. 'A Parallel Scheme for IDS'.

Zhou, Z., Chen, Z., Zhou, T., and Guan, X. (eds.) (2010) *2010 International Conference on Networking and Digital Society, ICNDS 2010*. 'The Study on Network Intrusion Detection System of Snort'.

Zhu, Y., Kang, N., Cao, J., Greenberg, A., Lu, G., Mahajan, R., Maltz, D., Yuan, L., Zhang, M., and Zhao, B. Y. (eds.) (2015) *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. 'Packet-Level Telemetry in Large Datacenter Networks': ACM.

Zhuang, Z., Luo, Y., Li, M., and Weng, C. (eds.) (2008) *Proceedings - 2008 IFIP International Conference on Network and Parallel Computing, NPC 2008*. 'A Resource Scheduling Strategy for Intrusion Detection on Multi-Core Platform'.

Appendix

Appendix 1. QoS Configuration

Section 1: Lab Task 1

Part 1

Globally Enabling QoS

SW3560 (config) # mls qos "***enable QoS"***

Enable VLAN-Based classification

Classification can be port-based or VLAN-based. To use the VLAN-based approach; enable VLAN-Based QoS on individual interfaces" port interfaces that bellowing in that VLAN":

SW3560 (config-if) #mls qos VLAN-Based
SW3560 (config) #interface Vlan 100
SW3560 (config-if) # service-policy input test ***"apply the policy to Switch Vlan Interface (SVI)"***
SW3560 (config-if) #exit
SW3560 (config) #interface FastEthernet 0/1
SW3560 (config-if) #mls qos Vlan-based ***"take the qos policy setting from SVI"***

Setting the port's CoS value

SW3560 (config-if) #mls qos CoS ***"value 0-7"***
SW3560 (config-if) #mls qos CoS override ***"that mean it will take default value"***

Setting the Ports trust state

SW3560 (config-if) #mls qos trust ?
SW3560 (config-if) # mls qos trust ip ***"trust IP traffic"***
SW3560 (config-if) # mls qos trust udp
SW3560 (config-if) # mls qos trust icmp
SW3560 (config-if) # mls qos trust tcp
SW3560 (config) #mls qos trust CoS

QoS Mapping

Once trusted an incoming marking, remark that frame/ packet based on a mapping table.

SW3560 (config) #mls qos trust cos pass-through dscp
SW3560 (config) #show mls qos maps CoS-DSCP
SW3560 (config) # mls qos map ?
SW3560 (config) # mls qos map CoS-DSCP 0 8 16 24 32 46 48 56
SW3560 (config) #exit
SW3560 #show mls qos map CoS-DSCP

SW3560 (config) # class-map match-any dscp_class
SW3560 (config)# match ip dscp 9
SW3560 (config-cmap)# exit
SW3560 (config) # class-map match-all vlan_class
SW3560 (config-cmap)# match vlan 100
Switch(config-cmap)# match class-map dscp_class

Importance notes:
You could classification/or recognize different type of traffic at layer 2 using MAC address or access control list (ACL) also you can classify traffic in access layer3 by using access control list (ACL).

Part 2

Classifying and Class map

SW3560 (config) #access-list 100 permit udp any any ***"you can add or not range 16384 32767 Cisco rules used UDP range form 16384 – 32767"***
SW3560 (config) #access-list 100 deny ip any any
SW3560 (config) #access-list 110 permit tcp any any
SW3560 (config) #access-list 110 deny ip any any
SW3560 (config) #access-list 120 permit icmp any any
SW3560 (config) #access-list 120 deny ip any any

SW3560 (config) #class map (inter-class map name(C-UDP, C-TCP, C-ICMP)).
SW3560 (config) #class map C-UDP
SW3560 (config-C-map) #match access-group 100 "match access list 100 with class map"
SW3560 (config) #class map C-TCP
SW3560 (config-C-map) #match access-group 110
SW3560 (config) #class map C-ICMP
SW3560 (config-C-map) #match access-group 120

Creating policy map

SW3560 (config) #policy-map (inter name of policy map (P-UDP, P-TCP, P-ICMP)).
SW3560 (config) #policy-map P-UDP
SW3560 (config-Pmap) #police 100000000 8000 exceed-action policed-dscp-transmit
SW3560 (config-Pmap) #no match exceed-action drop.
"100000000 is rate limit in bytes / s (bps), 8000 is not rate, it is a number of bytes".
SW3560 (config) #policy-map P-TCP
SW3560 (config-Pmap) #police 100000000 8000.
SW3560 (config) #policy-map P-ICMP
SW3560 (config-Pmap) #police 100000000 8000.

Applying policy map in interface

SW3560 (config) #interface FastEthernet 0/1
SW3560 (config-if) #service-policy input policy P-UDP.
SW3560 (config-if) # mls qos trust dscp
SW3560 (config-if) # mls qos dscp-mutation FastEthernet 0/1-mutation
SW3560 (config) #interface FastEthernet 0/2
SW3560 (config-if) #service-policy input policy P-TCP.
SW3560 (config-if) # mls qos trust dscp
SW3560 (config-if) # mls qos dscp-mutation FastEthernet 0/2-mutation
SW3560 (config) #interface FastEthernet 0/3
SW3560 (config-if) #service-policy input policy P-ICMP.
SW3560 (config-if) # mls qos trust dscp
SW3560 (config-if) # mls qos dscp-mutation FastEthernet 0/3-mutation

All interface 1, 2 and 3 inside VLAN-1
Notes:
We should focus about the following below:
Delay traffic inside switch.
Fragmentation traffic in it.

Part 3

Congestion management (queuing)

Priority queue configuration

To change the default input priority queue configuration

```
SW3560 (config) #mls qos srr-queue input priority-queue?
     <1-2> enter priority queue number [1-2]
SW3560 (config) #mls qos srr-queue input priority-queue 1?
    Bandwidth ingress priority queue bandwidth & at stack ring
SW3560 (config) # mls qos srr-queue input priority-queue 1 bandwidth?
    <0-40> enter bandwidth number [0-40]
SW3560 (config) # mls qos srr-queue input priority-queue 1 bandwidth 30?
     <cr>
SW3560 (config) # mls qos srr-queue input priority-queue 1 bandwidth 30
```

Here from default ingress priority queue is queue2, it has 10 percentage of priority queue. So from the command above, we change priority queue form q2 to q1 and then we change the bandwidth to 30 percent of the interface bandwidth.

To enable output of priority queuing on queue1 into the port you can use this command:

```
SW3560 (config-if) #priority-queue out
```

Congestion Avoidance

Queue sets

By default, all ports belong to queue set 1. However ports can be assigned to a second queue set with the following command:

```
SW3560 (config-if) #queue-set queue_set-id
SW3560 (config)# mls qos queue-set output qset-id threshold queue-id drop threshold1 drop-threshold2 reserved-threshold maximum-threshold.
SW3560 (config)# mls qos queue-set output qset-id buffers allocation1 allocation2 allocation3 allocation4.
SW3560 (config)# mls qos queue-set output 2 buffers 50 25 15 10
                                                    Port buffer space
SW3560 (config)# mls qos queue-set output 2 threshold 100 90 70 50
SW3560 (config)# interface fast 0/1
SW3560 (config-if)# queue-set 2
```

For queue set2, 50 percent a port's buffer space is allocated for queue1. 25 percent is allocated for queue2. 10 percent is allocated for queue3. 15 percent is allocated for queue4.Also queue set2, output queue 2 of 4 has its first drop threshold at 100 percent and its second drop threshold at 90 percent. 100 percent of queue2 is allocate buffer space is guaranteed to be available , if needed if queue2 needs more buffer space , it can barrow from a port's unused buffer space, up to a maximum of 200 percent of queue 2 is buffers allocation.

Mapping QoS Markings to an output queue and drop threshold

SW3560 (config) #mls qos srr-queue output [CoS-map / DSCP-map] queue queue-id threshold threshold-id qos {marking-1…. Qos-marking-8}

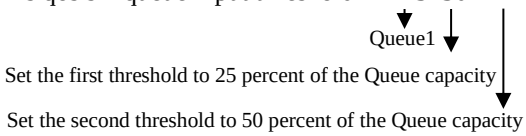
```
SW3560 (config) #mls qos srr-queue output CoS-map queue 1 threshold 1 0 1
SW3560 (config) #mls qos srr-queue output CoS-map queue 1 threshold 2 2 3
SW3560 (config) #mls qos srr-queue output CoS-map queue 2 threshold 1 4
SW3560 (config) #mls qos srr-queue output CoS-map queue 3 threshold 2 5
SW3560 (config) #mls qos srr-queue output CoS-map queue 4 threshold 2 6 7
                                                    Queue  Threshold  Threshold CoS
```

Part 4

Shaped Round Robin (SRR)
Bandwidth allocation for input queues

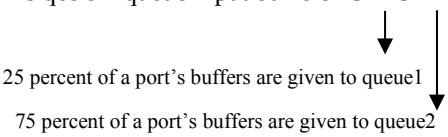
Notes: queue sets are not used for input queues.

SW3560 (config) #mls qos srr-queue input threshold queue-id threshold-precentage1 threshold-precentage2.
SW3560 (config) #mls qos srr-queue input threshold 1 25 50



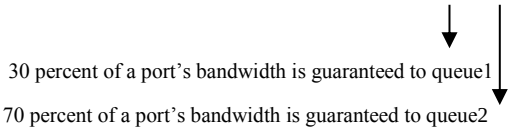
We can set the buffer allocation for input queues with the following command:

SW3560 (config) #mls qos srr-queue input buffers percentage1 precentage2
SW3560 (config) #mls qos srr-queue input buffers 25 75



To give different a mounts of bandwidth to input queues, we can use the following command:

SW3560 (config) #mls qos srr-queue input bandwidth weight1 weight2
SW3560 (config) #mls qos srr-queue input bandwidth 30 70



Bandwidth allocation for output queues (shared mode)

SW3560 (config-if) #srr-queue bandwidth share weight1 weight2 weight3 weight4
SW3560 (config-if) #srr-queue bandwidth share 30 20 25 25

30 percentages is relative weight for queue1, 20 percentages is relative weight for queue2, 25 percentages is relative weight for queue3, and then 25 percentages is relative weight for queue4.

Notes
TIP: the relative weight to not have to total 100. However, selecting values that do total 100 makes it easier to determine the bandwidth available to each queue.

So, 30 percentages it will be taken from the port's bandwidth for queue1, 20 percentages it will be taken from the port's bandwidth for queue2, 25 percentages it will be taken from the port's bandwidth for queue3 and 25 for queue4.

Determine the amount of bandwidth available to each output queue on interface FastEthernet.

SW3560 (config) # interface FastEthernet 0/2
SW3560 (config-if) #speed 100
SW3560 (config) # srr-queue bandwidth share 10 25 35 50

BW for Q1: [10/ (10+25+35+50)] =8.33 mbps
BW for Q2: [25/ (10+25+35+50)] =20.83 mbps
BW for Q3: [35/ (10+25+35+50)] =29.17 mbps
BW for Q4: [50/ (10+25+35+50)] =41.67 mbps
Total bandwidth (mbps) =8.33+20.83+29.17+41.67=100 mbps

Part 5

Bandwidth allocation for output queues (shaped mode)

SW3560 (config-if) #srr-queue bandwidth shape weight1 weight2 weight3 weight4
SW3560 (config-if) #srr-queue bandwidth share 50 50 0 0
50 inverse weight for queue1, 50 inverse weight for queue2, 0 shaping not applied for queue3, and 0 shaping not applied for queue4

Note:
TIP: if a queue is configured for both shaped and share mode, the shaped mode config will be applied, and share mode will be ignored.

Determine the amount of bandwidth limits applied to the output queues on interface FastEthernet.

SW3560 (config) # interface FastEthernet 0/3
SW3560 (config-if) #speed 100
SW3560 (config-f) # srr-queue bandwidth shape 30 0 0 0

BW limit for queue1:1/30*100 mbps=3.33 mbps
BW limit for queue2: no limit applied
BW limit for queue3: no limit applied
BW limit for queue4: no limit applied

Determine the amount of bandwidth guarantees or limits applied to the output queues on interface FastEthernet.

SW3560 (config) # interface FastEthernet 0/4
SW3560 (config-if) #speed 100
SW3560 (config-if) # srr-queue bandwidth shape 50 50 0 0
SW3560 (config-if) # srr-queue bandwidth share 100 100 40 20

The shaping config for a queue (i.e. non-zero value) overrides the sharing configuration.

BW limit for queue1 (mbps):1/50*100 mbps=20 mbps
BW limit for queue2 (mbps): 1/50*100 mbps=20 mbps
BW for queue3: [40/ (40+20)]*(100-20-20) mbps=40 mbps
BW for queue4: [20/ (40+20)]*(100-20-20) mbps=20 mbps
Total bandwidth (mbps) =20+20+40+20=100 mbps
Queue1 is guaranteed 20 mbps
Queue2 is guaranteed 20 mbps

Limiting Bandwidth on an output interface.

This command specifies the maximum amount of an interface's bandwidth that can be used for outgoing traffic, by default there is no limit(i.e a weight of 100)

SW3560 (config-if) # srr-queue bandwidth limit weight
SW3560 (config-if) # srr-queue bandwidth limit 85
Interface's outbound bandwidth is limit to 85precent of interface speed.

To confirm mls qos is enabled

SW3560# show mls qos.

To view a port's trust configuration

SW3560#show mls qos interface interface-id
SW3560# show mls qos interface FastEthernet 0/1

To view interface's policer configuration

SW3560#show mls qos interface interface-id policers.
SW3560#show mls interface FastEthernet 0/1 policers

To view a queue set's parameters:

SW3560#show mls qos queue-set

Section 2: Lap task2

SW3560 (config) # mls qos
SW3560 (config-if) #mls qos CoS
SW3560 (config-if) #mls qos CoS override

SW3560 (config-if) # mls qos trust IP
SW3560 (config-if) # mls qos trust UDP
SW3560 (config-if) # mls qos trust TCP
SW3560 (config-if) # mls qos trust ICMP

SW3560 (config) #mls qos trust CoS
SW3560 (config) #show mls qos maps CoS-DSCP
SW3560 (config) # mls qos map ?
SW3560 (config) # mls qos map CoS-DSCP 0 8 16 24 32 46 48 56
SW3560 (config) #mls qos trust dscp
SW3560 (config) #exit
SW3560 #show mls qos map CoS-DSCP

SW3560 (config) #access-list 100 permit ip any any / or
SW3560 (config) #access-list 100 permit ip-id range any any to ip-id range any any

SW3560 (config) #class map (class map id-name)
SW3560 (config) #class map id-name
SW3560 (config-C-map) #match access-group 100 "match access list 100 with class map"

SW3560 (config) #policy-map (inter name of policy map (id-policy name)).
SW3560 (config) #policy-map id-class map
SW3560 (config-Pmap) #police 25600000 8000 exceed-action policed-dscp-transmit.
SW3560 (config) #interface FastEthernet interface-id
SW3560 (config-if) #service-policy input policy policy-id.

SW3560 (config) #mls qos srr-queue input buffers percentage1 precentage2
SW3560 (config) #mls qos srr-queue input buffers 25 75

SW3560 (config) #mls qos srr-queue input threshold queue-id threshold-precentage1 threshold-precentage2.
SW3560 (config) #mls qos srr-queue input threshold 1 90 100
SW3560 (config) #mls qos srr-queue input bandwidth weight1 weight2
SW3560 (config) #mls qos srr-queue input bandwidth 50 50

SW3560 (config) #mls qos srr-queue input priority-queue queue-id
SW3560 (config) #mls qos srr-queue input priority-queue 2

SW3560 (config) #mls qos srr-queue input [CoS-map / DSCP-map] queue queue-id threshold threshold-id qos {marking-0…. Qos-marking-65} or {marking-1…. Qos-marking-8}

SW3560 (config) #mls qos srr-queue input dscp-map queue 1 threshold 1 0
SW3560 (config) #mls qos srr-queue input dscp-map queue 1 threshold 2 8
SW3560 (config) #mls qos srr-queue input dscp-map queue 1 threshold 3 16
SW3560 (config) #mls qos srr-queue input dscp-map queue 2 threshold 1 24
SW3560 (config) #mls qos srr-queue input dscp-map queue 2 threshold 2 32
SW3560 (config) #mls qos srr-queue input dscp-map queue 2 threshold 3 46
SW3560 (config) #mls qos srr-queue output dscp-map queue 1 threshold 2 8
SW3560 (config) #mls qos srr-queue output dscp-map queue 1 threshold 3 16
SW3560 (config) #mls qos srr-queue output dscp-map queue 2 threshold 2 32
SW3560 (config) #mls qos srr-queue output dscp-map queue 2 threshold 3 32
SW3560 (config) #mls qos srr-queue output dscp-map queue 2 threshold 2 46
SW3560 (config) #mls qos srr-queue output dscp-map queue 3 threshold 3 56
SW3560 (config) #mls qos srr-queue output dscp-map queue 4 threshold 1 0

SW3560 (config) #mls qos srr-queue output dscp-map queue 4 threshold 2 24
SW3560 (config) #mls qos srr-queue output dscp-map queue 4 threshold 3 48

SW3560 (config) # interface FastEthernet interface-id
SW3560 (config-if) #speed interface-speed –limit
SW3560 (config-if) #srr-queue bandwidth shape weight1 weight2 weight3 weight4
SW3560 (config-if) # srr-queue bandwidth shape x1 x2 x3 x4

Bandwidth limit for queue1: (1/x1)*interface-speed-limit mbps=bandwidth-limit for Q1 mbps
Bandwidth limit for queue2: (1/x2)*interface-speed-limit mbps=bandwidth-limit for Q2 mbps
Bandwidth limit for queue3: (1/x1)*interface-speed-limit mbps=bandwidth-limit for Q3 mbps
Bandwidth limit for queue4: (1/x1)*interface-speed-limit mbps=bandwidth-limit for Q4 mbps

SW3560 (config-if) #srr-queue bandwidth share weight1 weight2 weight3 weight4
SW3560 (config-if) # srr-queue bandwidth share y1 y2 y3 y4

Bandwidth limit for queue1: (y1/ (y1+y2+y3+y4)) =bandwidth-limit for Q1 mbps
Bandwidth limit for queue2: (y2/ (y1+y2+y3+y4)) =bandwidth-limit for Q2 mbps
Bandwidth limit for queue1: (y3/ (y1+y2+y3+y4)) =bandwidth-limit for Q3 mbps
Bandwidth limit for queue1: (y4/ (y1+y2+y3+y4)) =bandwidth-limit for Q4 mbps

SW3560 (config) # interface FastEthernet 0/1
SW3560 (config-if) #speed 100
SW3560 (config-if) # srr-queue bandwidth shape 88 88 0 0
SW3560 (config-if) # srr-queue bandwidth shape 100 100 60 40

For shaped mode:88 inverse weight for queue1, 88 inverse weight for queue2, 0 shaping not applied for queue3, and 0 shaping not applied for queue4.For share mode :10 percentages is relative weight for queue1, 100 percentages is relative weight for queue2, 60 percentages is relative weight for queue3, and then 40 percentages is relative weight for queue4.

BW limit for queue1 (mbps):1/88*100 mbps=1.13 mbps
BW limit for queue1 (mbps):1/88*100 mbps=1.13 mbps
BW for queue3: [60/ (60+40)]*(100-1.13-1.13) mbps=58.644 mbps
BW for queue4: [40/ (60+40)]*(100-1.13-1.13) mbps=39.096 mbps
Total bandwidth (mbps) =1.13+1.13+58.644+39.096=100 mbps

Section 3: Lab task 3

SW3560 (config) #access-list 101 permit UDP any any
SW3560 (config) #access-list 101 deny ip any any
SW3560 (config) #access-list 102 permit TCP any any
SW3560 (config) #access-list 102 deny ip any any
SW3560 (config) #access-list 103 permit ICMP any any
SW3560 (config) #access-list 103 deny ip any any
SW3560 (config) #access-list 104 permit ip any any

SW3560 (config) #class map C-UDP
SW3560 (config-C-UDP) #match access-group 101 ***"match access list 101 with class map"***
SW3560 (config-C-UDP) #exit

SW3560 (config) #class map C-TCP
SW3560 (config-C-UDP) #match access-group 102 ***"match access list 102 with class map"***
SW3560 (config-C-UDP) #exit

SW3560 (config) #class map C-ICMP
SW3560 (config-C-UDP) #match access-group 103 ***"match access list 103 with class map"***
SW3560 (config-C-UDP) #exit

SW3560 (config) #class map C-Other
SW3560 (config-C-UDP) #match access-group 104 "***match access list 104 with class map"***
SW3560 (config-C-UDP) #exit
Policy
SW3560 (config) #policy-map P-UDP
SW3560 (config) #policy-map C-UDP
SW3560 (config-Pmap) #police 13500000 8000 exceed-action policed-dscp-transmit
SW3560 (config-Pmap) #exit

SW3560 (config) #policy-map P-TCP
SW3560 (config) #policy-map C-TCP
SW3560 (config-Pmap) #police 13500000 8000 exceed-action policed-dscp-transmit
SW3560 (config-Pmap) #exit

SW3560 (config) #policy-map P-ICMP
SW3560 (config) #policy-map C-ICMP
SW3560 (config-Pmap) #police 13500000 8000 exceed-action policed-dscp-transmit
SW3560 (config-Pmap) #exit

SW3560 (config) #policy-map P-Other
SW3560 (config) #policy-map C-Other
SW3560 (config-Pmap) #police 100000000 8000 exceed-action policed-dscp-transmit
SW3560 (config-Pmap) #exit

SW3560 (config) #interface FastEthernet interface-id/ VLAN-id

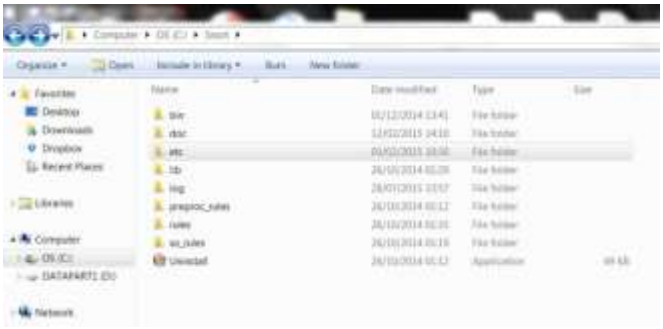
SW3560 (config) #interface FastEthernet 0/1
SW3560 (config-if) #service-policy input policy P-UDP
SW3560 (config-if) #exit
SW3560 (config) #interface FastEthernet 0/2
SW3560 (config-if) #service-policy input policy P-TCP
SW3560 (config-if) #exit
SW3560 (config) #interface FastEthernet 0/3
SW3560 (config-if) #service-policy input policy P-ICMP
SW3560 (config-if) #exit
SW3560 (config) #interface FastEthernet 0/4
SW3560 (config-if) #service-policy input policy P-Other
SW3560 (config-if) #exit.

Appendix 2. Installation and Configuration of Snort NIDPS.

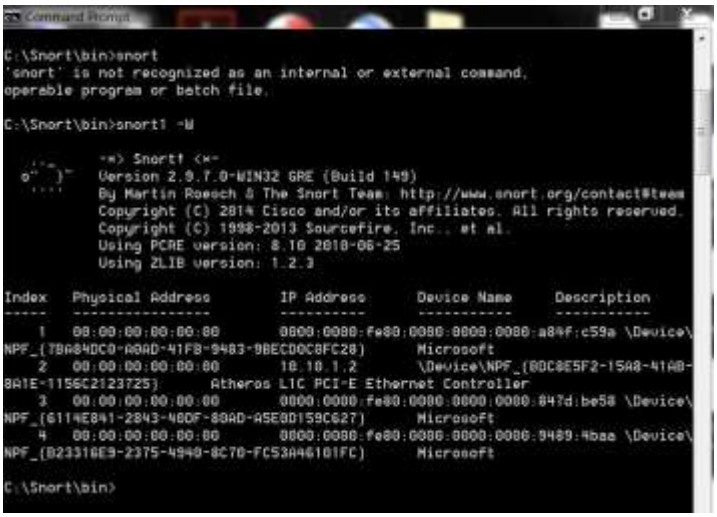
The following sections give instructions for installing and configuring Snort NIDPS for Windows, Linux OSs and virtual machines

Section 1: Windows

1. Download Snort, Snort rules and Winpcap tools from the following website " www.snort.org"
2. When you downloaded you will see the following structure on C: driver



3. Go to command prompt
4. C:>Snort –W



5. Go to C:\snort\etc\snort.config and then change some lines as the following :

From :
Path to your rules files (this can be a relative path)
Note for Windows users: You are advised to make this an absolute path,
such as: c:/snort/rules
var RULE_PATH c:/snort/rules
var SO_RULE_PATH c:/snort/so_rules
var PREPROC_RULE_PATH c:/snort/preproc_rules
If you are using reputation preprocessor set these
var WHITE_LIST_PATH c:/snort/rules
var BLACK_LIST_PATH c:/snort/rules

To :
such as: c:\snort\rules
var RULE_PATH c:\snort\rules
var SO_RULE_PATH c:\snort\so_rules

var PREPROC_RULE_PATH c:\snort\preproc_rules
var WHITE_LIST_PATH c:\snort\rules
var BLACK_LIST_PATH c:\snort\rules

From:
Configure default log directory for snort to log to. For more information see snort -h command line options (-l)
#config logdir: c:/snort/log
To:
#config logdir: c:\snort\log

From:
Dynamic Modules
##
path to dynamic preprocessor libraries
dynamicpreprocessor directory c:/snort/_dynamicpreprocessor
path to base preprocessor engine
dynamicengine c:/snort/snort_dynamicengine/engine.dll
path to dynamic rules libraries

To :
Dynamic Modules
##
path to dynamic preprocessor libraries
dynamicpreprocessor directory c:\snort\lib\snort_dynamicpreprocessor
path to base preprocessor engine
dynamicengine c:\snort\lib\snort_dynamicengine\sf_engine.dll
path to dynamic rules libraries
dynamicdetection directory c:\snort\lib\snort_dynamicrules

Add:
Include $RULE_PATH/local.rules in step7 list rule

Add:
whitelist $WHITE_LIST_PATH\white.list, \
blacklist $BLACK_LIST_PATH/black.list
At the send of step 5

6. Open local rule file and create your rules. The list of rules can be found it in the list below:

LOCAL RULES

- #alert udp any any -> any any (msg:" Malicious UDP Packets "; sid:1000002;)
- #drop ip any any -> any any (msg:" ip Header TEST"; sid:1000001;)
- # block ip any any -> any any (msg:" ip Header TEST"; sid:10000001;)
- # alert tcp any any -> any any (msg:" TCP Packets Generatore TEST"; sid:1000002;)
- # alert icmp any any -> any any (msg:" ICMP Header Test "; sid:1000003;)
- # alert ip any any -> any any (msg:" udp flooder TEST"; sid:1000004;)
- # alert ip any any -> any any (msg:" ip Packets Generatore TEST"; sid:1000005;)
- # alert udp any any -> any any (msg:" detect a specific time to live value of ip header"; ttl: 128;
    - sid:1000006;)
- # alert ip any any -> any any (msg:" ip flooder TTL=128 TEST"; ttl: 128; sid:1000007;)
- # alert udp any any -> any any (msg:" UDP flooder TTL=128 TEST"; ttl: 128; content: "abcdef"; sid:1000009;)
- # alert udp any any -> any any (msg:" UDP flooder TTL=128 TEST"; ttl: 128; content: "|72 7D 98 8D|"; sid:10000010;)
- # alert udp any any -> any any (msg:" start resrach for the word abcdef after 100 bytes from the start of the
    - data"; ttl: 128; content: "abcdef"; offset:100; sid:10000011;)
- # alert udp any any -> any any (msg:" start resrach for the word abcdef between characters4 and 100 bytes of the data"; ttl: 128; content: "abcdef"; offset: 4; depth: 100; sid:10000012;)
- # # alert udp any any -> any any (msg:" abc work match"; ttl: 128; content-list: "abc"; sid:10000014;)
- # alert ip any any -> any any (msg:" data size of an ip packets"; ttl: 128; dsize: <30000; sid:10000015;)
- # alert udp any any -> any any (msg:" data size of an udp packets"; ttl: 128; dsize: <30000; sid:10000016;)
- # alert icmp any any -> any any (msg:" check icmp-id field >oms ";icmp_id: 789; sid:10000017;)
- # alert IP any any -> any any (msg:" Check if the source and destination ip address are same "; sameip;
    - sid:10000018;)
- # alert tcp any any -> any any (msg:" Check sequence number "; seq: 1; sid:10000019;)
- # # Alert udp any any -> any any (msg:" check UDP-ID field"; id:789; Sid: 1000005 ;)

- # Alert icmp any any -> any any (msg:" check ICMP-ID field"; icmp_id:789; Sid: 1000005 ;)
- # Alert udp any any -> any any (msg:" Detect Malicious packets (Check Time To Live value in udp header)"; ttl: 128; Sid: 1000006 ;)

- # Alert ip any any -> any any (msg:" Check Time To Live value in ip header"; ttl: 128; Sid: 1000006 ;)
- # Alert udp any any -> any any (msg:" Detect Malicious packets (Check or to find data pattern inside
  - packets) "; ttl:128; content:"abcdef"; Sid:1000007;)
- # Alert udp any any -> any any (msg:" Detect Malicious packets (Check hexadecimal characters in side
  - data) "; ttl:128; content:"|61 62 63 64|"; Sid:1000008 ;)
- # Alert udp any any -> any any (msg:" Detect Malicious packets (Check data size for packets) "; ttl:128;
  - dsize:<660000; Sid:1000009 ;)
- # Alert udp any any -> any any (msg:" Start research for the word "abcdef" after 1 bytes from the start of data "; ttl:128; content:"abcdef"; offset:1; Sid:10000010 ;)
- # Alert udp any any -> any any (msg:" Detect Malicious packets (Start research for the word "abcdef"
  - between characters 1 and 100 bytes of the data) "; ttl:128; content:"abcdef"; offset:1; depth:100; Sid:10000011 ;)
- # Alert udp any any -> any any (msg:" Start research for the word "abcdef" after 100 bytes from the start of data "; ttl:128; content:"abcdef"; offset:4; Sid:10000010 ;)

Section 2: Linux

1. Update the system
   # apt-get update
   # apt-get install openssh-server
   # reboot
2. Install ethtool tool
   # apt-get install ethtool
3. Install build-essential tool
   # apt-get install -y build-essential
4. Install Snort prerequisites
   Install libpcap-dev, libpcre3-dev, zlib1g-dev and libdumbnet-dev packages
   # apt-get install -y libpcap-dev
   # apt-get install libpcre3-dev
   # apt-get install -y libdumbnet-dev
   # apt-get install zlib1g-dev
5. Install Snort DAQ Prerequisites
   bison and flex are the requirement for Snort DAQ installation
   # apt-get install bison flex
6. Create a directory to install tar packages of snort and Snort DAQ
   # mkdir /home/snort/snort_src
7. Change working directory to newly created directory.
   # cd /home/snort/snort_src/
8. Download and install DAQ
   # wget https://www.snort.org/downloads/snort/daq-2.0.6.tar.gz
9. Install the Package
   # tar -xvf daq-2.0.6.tar.gz
   # cd daq-2.0.6
   # cd daq-2.0.6
   # ./configure
   # makes root@snort:/home/snort/snort_src/daq-2.0.6# make install
10. Install Snort in same directory
    # wget https://www.snort.org/downloads/snort/snort-2.9.7.5.tar.gz
11. Extract and Install the snort package
    # gunzip snort-2.9.7.5.tar.gz
    # tar -xvf snort-2.9.7.5.tar
    # cd snort-2.9.7.5
    # ./configure --enable-sourcefire
    # make
    # make install
    # ldconfig
12. Create a Soft Link for Snort binary

ln -s /usr/local/bin/snort /usr/sbin/snort

13. Verify your Snort is installed correctly or not
    # snort -V
14. Configure Snort for NIDS Mode
    Create following Directories
    # mkdir /etc/snort
    # mkdir /etc/snort/rules
    # mkdir /etc/snort/preproc_rules
    # touch /etc/snort/rules/white_list.rules
    # touch /etc/snort/rules/black_list.rules
    # touch /etc/snort/rules/local.rules
15. Create Log Directory for snort
    # mkdir /var/log/snort
16. Create a Directory for snort Dynamics rules
    # mkdir /usr/local/lib/snort_dynamicrules
17. Change permissions
    # chmod -R 5775 /etc/snort/
    # chmod -R 5775 /var/log/snort/
    # chmod -R 5775 /usr/local/lib/snort
    # chmod -R 5775 /usr/local/lib/snort_dynamicrules/
18. Copy *.conf and *.map files from snort download directory to /etc/snort
    # cp /home/snort/snort_src/snort-2.9.7.5/etc/*.conf* /etc/snort/
    # cp -v /home/snort/snort_src/snort-2.9.7.5/etc/*.map* /etc/snort/
19. Configure /etc/snort/snort.conf
    Before editing snort.conf get the backup of that file first
    # cp /etc/snort/snort.conf /etc/snort/snort.conf_orig
20. Give following Command
    # sed -i 's/include \$RULE_PATH/#include \$RULE_PATH/' /etc/snort/snort.conf
    Note:Above Command will comment all rulesets which we will edit line by line
21. Go to line 45 of /etc/snort/snort.conf, edit to make like below
    ipvar HOME_NET 10.0.0.0/8
    ipvar EXTERNAL_NET !$HOME_NET
22. Go to line 104 and put following entries
    var RULE_PATH /etc/snort/rules
    var SO_RULE_PATH /etc/snort/so_rules
    var PREPROC_RULE_PATH /etc/snort/preproc_rules
    var WHITE_LIST_PATH /etc/snort/rules
    var BLACK_LIST_PATH /etc/snort/rules
23. To enable local rules go to line 551 and uncomment following line
    ##include $RULE_PATH/local.rules
24. Save and Quit
25. Now Download Community rules from following link
    https://www.snort.org/downloads/community/community-rules.tar.gz
    Extract these rules and copy to /etc/snort/rules.
26. Test the configuration
    # snort -T -c /etc/snort/snort.conf

Section 3: For a virtual machine

1. Download and run Ubuntu V 14.04.3 LTS in virtual machine
2. Go to terminal
3. :$ ifconfig
4. :$//installing prerequisite for compliny snort//
5. :$sudo apt-get install flex bison build-essential checkinstall libpcap-dev libnet1.dev libpcre3-dev libmysqlclient15.dev libnetfilter-queue-dev iptables-dev
6. :$do you want to continue [y/n] y
7. :$wget http://libdnet.google.com/files/libdnet-1.12.tgz
8. :$ls
9. :$tar xvf2 libdnet-1.12.tg2
10. :$ls
11. :$cd libdnet-1.121
12. :$~/ libdnet-1.12 $ ./configure “CFLAGS—FPIC”
13. :$make
14. :$sudo checkinstall
15. :$ do you want to list them [y/n]: n
16. :$Should i exclude ……..[y/n]:y
17. :$sudo dpkg –I libnet-1.12-1_amd64.deb
18. :$sudo ln –s /usr / local / lib/ libnet.1.0.1 /usr/lib/libdnet.1
19. :$//” download “ snort/2//
20. :$//”go to internet browser and them open www.snort.org>>download snort>>source>>daq. tar.g2>>snort.V.targ2”//
21. :$cd
22. :$ls
23. Cd downloads
24. ~/downloads$ ls
25. //” you will find DAQ file and Snort file.//
26. :$tar xvfz dag-.V.tar.g2
27. :$ls
28. :$cd daq-2.0.2/
29. :$./configure
30. :$make
31. :$sudo checkinstall
32. :$Do you want to list them[y/n]:n
33. :$Sudo dpkg –I dag-2.V.deb
34. :$ls
35. :$cd
36. :$cd downloads
37. :$tar xvfvz snort .V. tar.g2
38. :$ls
39. :$cd snort
40. :$./configure
41. :$make
42. :$sudo checkinstall
43. :$sudo dpkg –I snort.V.deb
44. :$sudo in –s /usr/local/bin/snort /usr/sbin/snort
45. :$sudo ldconfig –v
46. :$snort –v
47. :$// netx step to configure snort and download snort rules//
48. :$// for security we requmanded to create seprate linex user//
49. :$sudo groupadd snort
50. :$sudo useradd snort
51. :$sudo useradd snort –d/var/log /snort –s/sbin/nologin –c snort_ids –g snort
52. :$sudo mkdir /var/log/snort
53. :$sudo chown snort:snort /var/log/snort
54. :$//go to snort rule//
55. :$//www.snort.org>>rule>>sign in>>get rules>>snort rule file>>save//
56. :$cd Downloads

57. :$sudo mkdir /etc/snort
58. :$Sudo tar xvfvz snortrules…. Tar.g2 –c /etc/snort
59. :$sudo touch /etc/snort/rules/white_list.rules/etc/snort/rules/black_list.rules
60. :$sudo mkdir /usr/local/lib/snort/snort_dynamicrules.
61. :$sudo chown –R snort:snort /etc/snort/
62. :$sudo mv /etc/snort/etc/* /etc/snort
63. :$// configure snort configuration file//
64. :$sudo pico /etc/snort/snort.conf
65. :$// file snort.config is oppend >> changed >>
66. *step#1
67. #Ipvar home_net ANY [YOUR NETWORK]EX[10.0.0.0/8]
68. #Ipvar EXTERNAL_NET any >> [!$HOME_NET]
69. #VAR rue.path. /etc/snort/rules
70. # /etc/snort/so_rules
71. # /etc/snort/pre_rules
72. #var wite_list_path /etc/snort/rules
73. /ect/snort/rules
74. #Close snort.config file
75. :$sudo –T –I eth0 –A snort –g snort –c /etc/snort/snort.config
76. :$// install tcp replay//
77. :$wget https://dl.dropboxusercontent.com/u/98306176/captures/smallflows.pcap
78. :$wget http://download s.sourceforge.net/project/tcpreplay/tcpreplay/4.0.0/tcpreplay-4.0.0tar.gz
79. :$download libcap-1.6.2 tarz , “tcpdump-4.6.2”
80. :$tar xvzf libcap.16.2 tar.g2
81. :$cd libcap-1.6.2
82. :$./configure; make; sudo make install
83. :$tar xvzf tcpreplay-4.0.0
84. Cd tcpreplay
85. :$./configure; make ; sudo make install
86. :$./sudo make test
87. :$tcpreplay –V
88. :$sudo tcpreplay –I eth –t –k small flows.pcap

Appendix 3. Terms and Expressions (Abbreviations)

ARP: Address Resolution Protocol.

ARP: Address Resolution Protocol.

ASA: Adaptive Security Appliance.

API: Application Program Interface.

ATM: Asynchronous Transfer Mode.

ASIC: Application-Specific Integrated Circuit.

ACL: Access Control List.

BVI: Bridge-group Virtual Interface.

Bps: Byte per Second.

Bpms: Byte per millisecond.

CPU: Central processing Unit.

CSI: Crime Scene of Investigation.

CGI: Common Gateway interface.

CDP: Cisco Discovery Protocol.

CoS: Class of Service.

CMF: Constrained Multicast Flooding.

DoS: Denial of Service.

DDoS: Distributed Denial of Service.

DiffServ: Differentiated Service.

DSCP: Differentiated Service Code Point.

EIGRP: Enhanced Interior Gateway Routing Protocol.

Eth: Ethernet interface.

Frag: Fragmented Packets.

FEC: Fast Ethernet Channel.

FBI: Federal Bureau of Investigation.

GrIDPS: Graph-based Intrusion Detection and Prevention System.

GUI: Graphical User Interface.

GEC: Gigabit Ethernet Channel.

GPU: Graphics Processor Unit.

GB: Gigabyte.

Gb: Gigabit.

Gbps: Gigabit per second.

HIDPS: Host-based Intrusion Detection and Prevention System.

HTTP: Hypertext Transfer Protocol.

HSRP: Host Standby router Protocol.

ID: Intrusion Detection.

IP: Intrusion Prevention.

IDS: Intrusion Detection System.

IDPS: Intrusion Detection and Prevention System.

IPS: Intrusion Prevention System.

ICMP: Internet Control Massage Protocol.

IOS: Interface Operating system.

IPX: Internetwork Packet Exchange.

ISL: Inter-Switch Link.

I/O: Input and Output.

KBps: Kilobytes per second.

LAN: Local Area Network.

Libpcap: Library Packets Capture.

MIB: Management Information Bases.

MIDeA: Multi-parallel IDS Architecture.

MB: Megabyte.

Mb: Megabit.

Ms: Millisecond.

mSec: Microsecond.

MHz: Megahertz.

NID: Network Intrusion Detection.

NIP: Network Intrusion Prevention.

NIDPS: Network-based Intrusion Detection and Prevention System.

NAPI: New Application Program Interface.

NIC: Network Interface Controller.

OSs: Operating Systems.

OSC: Operating System Compatibility.

POP: Post Office Protocol.

PPP: Point-to-Point Protocol.

PDP: Policy Decision Point.

Pcap: Packet Capture.

PCRE: Perl-Compatible Regular Expressions.

PBR: Policy-Based routing.

PEM: Parallel Exact Matching.

QoS: Quality of Service.

OSPF: Open Shortest Path First.

OPI: Open System Interconnection.

RIP: Routing Information Protocol.

RMON: Remote MONitoring.

SMB: Server Massage Block.

SNMP: simple Network Management Protocol.

SLIP: Serial-Line internet Protocol.

SQL: Structured Query Language.

SONET: Synchronous Optical Network.

SSH: Secure Shell.

SSL: Secure Socket Layer.

SVI: Switch Virtual Interface.

SRR: Share or Shape Round Robin.

SRA: Snort Rule Accelerator.

TCP: Transmission Control Protocol.

ToS: Type of Service.

TTL: Time-To-Live.

UDP: User Datagram Protocol.

VPN: Virtual Private Network.

VIA: Virtual Internet Access.

VLAN: Virtual Local Network Interface.

WISR: Worldwide Infrastructure Security Report.

WTD: Weighted Tail Drop.

WinPcap: Windows Packets Capture.

XML: Extensible Mark-up Language.

λ: Traffic (packets) speed rate.

β: Buffer Speed rate.

λ_d: Drop packets rate.

λ_o: Outstanding packets rate.

β_{ex_i}: Output buffer rate for interface i.

$\beta_{in_{i_j}}$: Ingress queue (j) buffer rate for interface i.

$\beta_{ex_{i_j}}$: Egress queue (j) buffer rate for interface i.

λ_{in}: The arrival traffic (packets) rate.

λ_{in_i}: The arrival traffic (packets) rate for interface i.

λ_{out}: The output rate for output links (interface).

λ_{out_i}: The output traffic (packets) rate for output link (interface i).

α: The maximum rate of each buffer.

μ: Group of bytes per seconds.

β_k: Kernel buffer rate.

$\alpha_{out_{i_j}}$: The output packets rate for queue j in interface i.

β_{svi}: SVI memory pool rate.

β_{ex}: The rate for all egress buffers together.

β_{sw_i}: Switch memory common pool rate.

Appendix 4: Ethical Approval form

Low Risk Research Ethics Approval Checklist

Applicant Details

Name: Waleed A A Ahmed Bul'ajoul	**E-mail:**bulajouw@uni.coventry.ac.uk
Department: Faculty of Engineering and Computing	**Date:**11/01/2016
Course: PhD. Computing(Network and Security)	**Title of Project:** Performance of Network Intrusion Detection and Prevention System in High Speed Environments

Project Details

Intrusion Detection systems for high speed environments Improve Performance of Network Intrusion Detection and Prevention Systems (NIDPS) Through QoS and Parallel technologies

Participants in your research

1. Will the project involve human participants?	Yes	No ✓

If you answered **Yes** to this questions, this may **not** be a low risk project.

- If you are a student, please discuss your project with your Supervisor.
- If you are a member of staff, please discuss your project with your Faculty Research Ethics Leader or use the Medium to High Risk Ethical Approval or NHS or Medical Approval Routes.

Risk to Participants

2. Will the project involve human patients/clients, health professionals, and/or patient (client) data and/or health professional data?	Yes	No ✓
3. Will any invasive physical procedure, including collecting tissue or other samples, be used in the research?	Yes	No ✓
4. Is there a risk of physical discomfort to those taking part?	Yes	No ✓
5. Is there a risk of psychological or emotional distress to those taking part?	Yes	No ✓
6. Is there a risk of challenging the deeply held beliefs of those taking part?	Yes	No ✓
7. Is there a risk that previous, current or proposed criminal or illegal acts will be revealed by those taking part?	Yes	No ✓
8. Will the project involve giving any form of professional, medical or legal advice, either directly or indirectly to those taking part?	Yes	No ✓

If you answered **Yes** to **any** of these questions, this may **not** be a low risk project.

- If you are a student, please discuss your project with your Supervisor.
- If you are a member of staff, please discuss your project with your Faculty Research Ethics Leader or use the Medium to High Risk Ethical Approval or NHS or Medical Approval Routes.

Risk to Researcher

9. Will this project put you or others at risk of physical harm, injury or death?	Yes	No ✓
10. Will project put you or others at risk of abduction, physical, mental or sexual abuse?	Yes	No ✓
11. Will this project involve participating in acts that may cause psychological or emotional distress to you or to others?	Yes	No ✓
12. Will this project involve observing acts which may cause psychological or emotional distress to you or to others?	Yes	No ✓
13. Will this project involve reading about, listening to or viewing materials that may cause psychological or emotional distress to you or to others?	Yes	No ✓
14. Will this project involve you disclosing personal data to the participants other than your name and the University as your contact and e-mail address?	Yes	No ✓
15. Will this project involve you in unsupervised private discussion with people who are not already known to you?	Yes	No ✓
16. Will this project potentially place you in the situation where you may receive unwelcome media attention?	Yes	No ✓
17. Could the topic or results of this project be seen as illegal or attract the attention of the security services or other agencies?	Yes	No ✓
18. Could the topic or results of this project be viewed as controversial by anyone?	Yes	No ✓

If you answered **Yes** to **any** of these questions, this is **not** a low risk project. Please:

- If you are a student, discuss your project with your Supervisor.
- If you are a member of staff, discuss your project with your Faculty Research Ethics Leader or use the Medium to High Risk Ethical Approval route.

Informed Consent of the Participant

19. Are any of the participants under the age of 18?	Yes	No ✓
20. Are any of the participants unable mentally or physically to give consent?	Yes	No ✓
21. Do you intend to observe the activities of individuals or groups without their knowledge and/or informed consent from each participant (or from his or her parent or guardian)?	Yes	No ✓

If you answered **Yes** to **any** of these questions, this may **not** be a low risk project. Please:

- If you are a student, discuss your project with your Supervisor.
- If you are a member of staff, discuss your project with your Faculty Research Ethics Leader or use the Medium to High Risk Ethical Approval route.

Participant Confidentiality and Data Protection

22. Will the project involve collecting data and information from human participants who will be identifiable in the final report?	Yes	No ✓
23. Will information not already in the public domain about specific individuals or institutions be identifiable through data published or otherwise made available?	Yes	No ✓
24. Do you intend to record, photograph or film individuals or groups without their knowledge or informed consent?	Yes	No ✓
25. Do you intend to use the confidential information, knowledge or trade secrets gathered for any purpose other than this research project?	Yes	No ✓

If you answered **Yes** to **any** of these questions, this may **not** be a low risk project:

- If you are a student, discuss your project with your Supervisor.
- If you are a member of staff, discuss your project with your Faculty Research Ethics Leader or use the Medium to High Risk Ethical Approval or NHS or Medical Approval routes.

Gatekeeper Risk

26. Will this project involve collecting data outside University buildings?	Yes	No ✓
27. Do you intend to collect data in shopping centres or other public places?	Yes	No ✓
28. Do you intend to gather data within nurseries, schools or colleges?	Yes	No ✓
29. Do you intend to gather data within National Health Service premises?	Yes	No ✓

If you answered **Yes** to **any** of these questions, this is **not** a low risk project. Please:

- If you are a student, discuss your project with your Supervisor.
- If you are a member of staff, discuss your project with your Faculty Research Ethics Leader or use the Medium to High Risk Ethical Approval or NHS or Medical Approval routes.

Other Ethical Issues

30. Is there any other risk or issue not covered above that may pose a risk to you or any of the participants?	Yes	No ✓
31. Will any activity associated with this project put you or the participants at an ethical, moral or legal risk?	Yes	No ✓

If you answered **Yes** to these questions, this may **not** be a low risk project. Please:

- If you are a student, discuss your project with your Supervisor.
- If you are a member of staff, discuss your project with your Faculty Research Ethics Leader.

Principal Investigator Certification

If you answered **No** to **all** of the above questions, then you have described a low risk project. Please complete the following declaration to certify your project and keep a copy for your record as you may be asked for this at any time.

Agreed restrictions to project to allow Principal Investigator Certification

Please identify any restrictions to the project, agreed with your Supervisor or Faculty Research Ethics Leader to allow you to sign the Principal Investigator Certification declaration.

Participant Information Leaflet attached.
Informed Consent Forms attached.
Risk Assessment Form attached.

Principal Investigator's Declaration

Please ensure that you:

- Tick all the boxes below and sign this checklist.
- Students must get their Supervisor to countersign this declaration.

I believe that this project **does not require research ethics approval.** I have completed the checklist and kept a copy for my own records. I realise I may be asked to provide a copy of this checklist at any time.	✓
I confirm that I have answered all relevant questions in this checklist honestly.	✓
I confirm that I will carry out the project in the ways described in this checklist. I will immediately suspend research and request a new ethical approval if the project subsequently changes the information I have given in this checklist.	✓

Signatures

If you or your supervisor does not have electronic signatures, please type your name in the signature space. An email sent from the Supervisor's University inbox will be accepted as having been signed electronically.

Principal Investigator

Signed..... [redacted] .....(Principal Investigator or Student)

Date: 11.01.2016

Students storing this checklist electronically must append to it an email from your Supervisor confirming that they are prepared to make the declaration above and to countersign this checklist. This email will be taken as an electronic countersignature.

Student's Supervisor

Countersigned..... [redacted] .....(Supervisor)

Date ..... 11/1/16 .....

I have read this checklist and confirm that it covers all the ethical issues raised by this project fully and frankly. I also confirm that these issues have been discussed with the student and will continue to be reviewed in the course of supervision.